▼ AMSTRAD ▼ SPECTRUM ▼ COMMODORE ▼ IBM ▼ MSX ▼

Informática y programación

32

PASO A PASO



PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Informática y programación 32

PASO A PASO



PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Una publicación de

EDICIONES SIGLO CULTURAL, S.A.

Director-editor:
RICARDO ESPAÑOL CRESPO.

Gerente:
ANTONIO G. CUERPO.

Directora de producción:
MARIA LUISA SUAREZ PEREZ.

Directores de la colección:
MANUEL ALFONSECA, Doctor Ingeniero de Telecomunicación y Licenciado en Informática.
JOSE ARTECHE, Ingeniero de Telecomunicación.

Diseño y maquetación:
BRAVO-LOFISH.

Fotografía:
EQUIPO GALATA.

Dibujos:
JOSE OCHOA

TECNICAS DE PROGRAMACION: Manuel Alfonseca, Doctor Ingeniero de Telecomunicación y Licenciado en Informática. TECNICAS DE ANALISIS: José Arteche, Ingeniero en Telecomunicación. LENGUAJE MAQUINA 8086: Juan Rojas Licenciado en Ciencias Físicas e Ingeniero Industrial. PASCAL: Juan Ignacio Puyol, Ingeniero Industrial. PROGRAMAS (educativos, de utilidad, de gestión y de juegos): Francisco Morales, Técnico en Informática y colaboradores. Coordinador de AULA DE INFORMATICA APLICADA (AIA): Alejandro Marcos, Licenciado en Ciencias Químicas. BASIC: Esther Maldonado, Diplomada en Arquitectura. INFORMATICA BASICA: Virginia Muñoz, Diplomada en Informática. LENGUAJE MAQUINA Z-80: Joaquín Salvachúa, Diplomado en Telecomunicación y José Luis Tojo, Diplomado en Telecomunicación. LENGUAJE MAQUINA 6502: (desde el tomo 5): Juan José Gómez, Licenciado en Química. LOGO: Cristina Manzanera, Licenciada en Informática. APLICACIONES: Jorge Thomas, Técnico en Informática. OTROS LENGUAJES (ADA): Joaquín Salvachúa, Diplomado en Telecomunicación y José Luis Tojo, Diplomado en Telecomunicación.

Ediciones Siglo Cultural, S.A.

Dirección, redacción y administración:
Pedro Teixeira, 8, 2.ª planta. Teléf. 455 09 99. 28020 Madrid.

Publicidad:
Gofar Publicidad, S.A. Benito de Castro, 12 bis. 28028 Madrid.

Distribución en España:
COEDIS, S.A. Valencia, 245. Teléf. 215 70 97. 08007 Barcelona.
Delegación en Madrid: Serrano, 165. Teléf. 411 11 48.

Distribución en Ecuador: Muñoz Hnos.

Distribución en Perú: DISELPESA.

Distribución en Chile: Alfa Ltda.

Importador exclusivo Cono Sur:
CADE, S.R.L. Pasaje Sud América, 1532. Teléf.: 21 24 64.
Buenos Aires - 1.290. Argentina.

ISBN del tomo: 84-7688-183-5
ISBN de la obra: 84-7688-068-7

Fotocomposición:
ARTECOMP, S.A. Albarracín, 50. 28037 Madrid.

Imprime:
MATEU CROMO. Pinto (Madrid).

Depósito legal: M-5-677-1987
Printed in Spain - Impreso en España.

Suscripciones y números atrasados:
Ediciones Siglo Cultural, S.A.
Pedro Teixeira, 8, 2.ª planta. Teléf. 259 73 31. 28020 Madrid.

Enero, 1988
P.V.P. Canarias: 335,–.

INDICE

4 INFORMATICA BASICA

8 MAQUINA 6502

11 PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

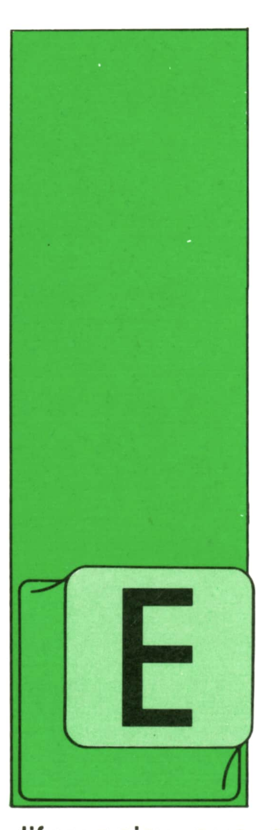
23 TECNICAS DE ANALISIS

25 TECNICAS DE PROGRAMACION

29 LOGO

33 PASCAL

38 OTROS LENGUAJES



INFORMATICA BASICA

EXPLOTACION

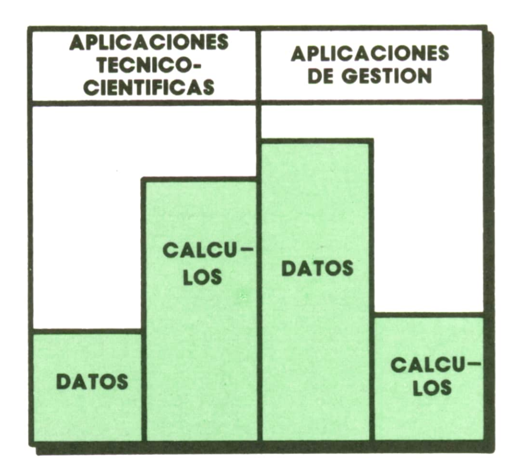


Introducción

EN los temas anteriores explicábamos con detalle cuáles son los pasos a seguir cuando se necesita realizar un proyecto informático. En este capítulo vamos a ver las diferencias que existen entre dos tipos muy comunes de aplicaciones.

Se puede hacer una clara distinción entre aplicaciones, las que están más orientadas a campos técnico-científicos y las típicas aplicaciones de gestión.

Los problemas del primer tipo se caracterizan porque el ordenador ejecuta muchas operaciones y cálculos, sobre un número relativamente pequeño de datos. Las aplicaciones de gestión suelen tratar muchos más datos, pero los procesos tienen pocas operaciones que realizar. Veamos cada caso separadamente.





Porcentaje de datos y cálculos en cada uno de los tipos de aplicaciones.

Aplicaciones técnicas

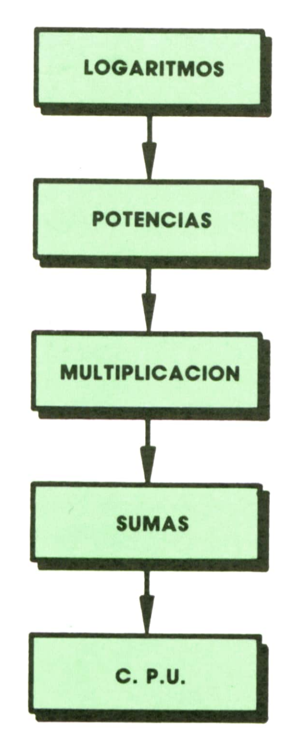
Como ya sabemos, las primeras fases en todo proyecto informático son: análisis de viabilidad, análisis funcional y análisis orgánico.

En aplicaciones dirigidas a resolver problemas técnicos las dos primeras fases no suelen tener tanta importancia como en las aplicaciones de gestión. Se trata de obtener resultados en función de unos datos de entrada; por tanto, cuando se haya determinado dicha función, el análisis está finalizado.

La mayor complicación en problemas de este tipo está a la hora de programar. Una vez realizado el análisis y determinadas las operaciones que hay que efectuar sobre los datos, hay que adaptar estas funciones a las operaciones que el ordenador es capaz de ejecutar.

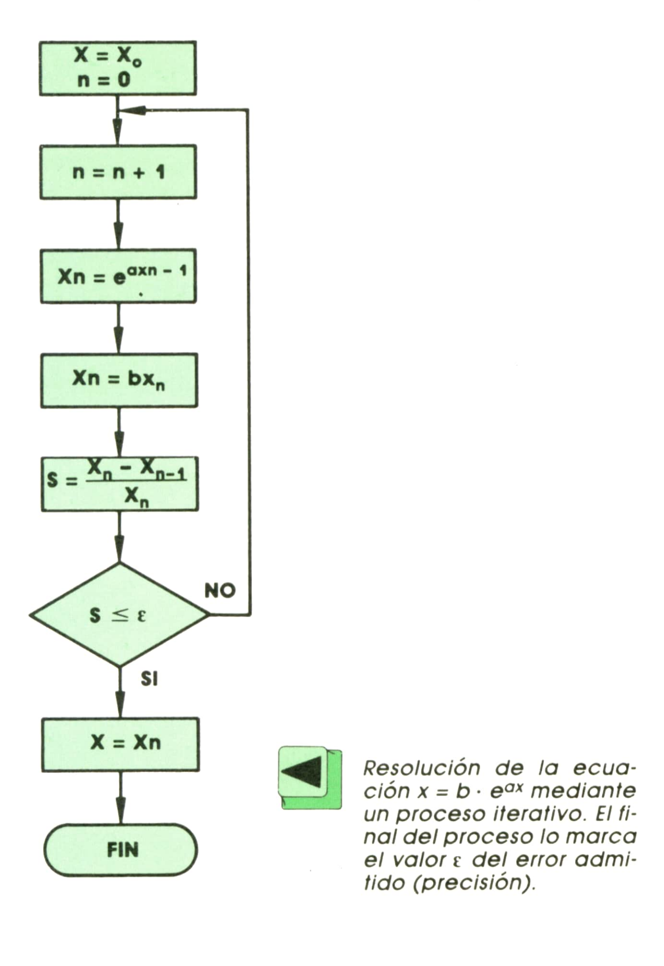
Sólo pueden realizarse ciertas operaciones básicas, cuyo número depende del ordenador utilizado. Los más sencillos pueden ejecutar operaciones de suma, resta, movimiento y negación de datos; otros más potentes han incorporado a su repertorio de instrucciones la multiplicación y división.

Teniendo en cuenta que en un problema técnico los cálculos son mucho más complicados que simples sumas o restas, la labor de los analistas consiste en buscar métodos de análisis numéricos mediante los cuales una operación complicada queda reducida a operaciones básicas. Esto requiere una fuerte preparación matemática del analista.



Es necesario simplificar las operaciones a realizar de manera que el ordenador pueda ejecutarlas.

Actualmente son más numerosas las firmas que ofrecen al programador rutinas estándar que resuelven estos problemas,

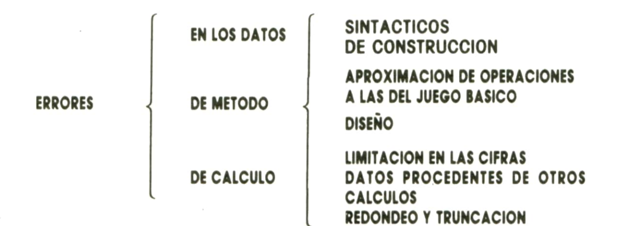


Resolución de la ecuación $x = b \cdot e^{ax}$ mediante un proceso iterativo. El final del proceso lo marca el valor ε del error admitido (precisión).

proporcionando la correspondencia funcional requerida.

Derivado del problema que existe al disponer de tan pocas operaciones ejecutables por el ordenador, está la necesidad de realizar multitud de iteraciones.

Esto en cuanto a tiempo no es demasiado importante, teniendo en cuenta la velocidad de cálculo que poseen los ordenadores. El verdadero problema está en los errores de cálculo que se aparece como consecuencia de la limitación de espacio de almacenamiento de los datos. Hay veces en las que las cantidades que se obtienen tiene un número tan grande de cifras que no pueden almacenarse en el espacio reservado para ellos. Esto se soluciona «truncando» o redondeando los números. Cuando esta operación se realiza una vez, el error puede no ser apreciable, pero si el número de iteraciones que se realizan es muy grande el error se irá incrementando en cada repetición, por lo que llegará un momento en que la solución obtenida no sea válida.



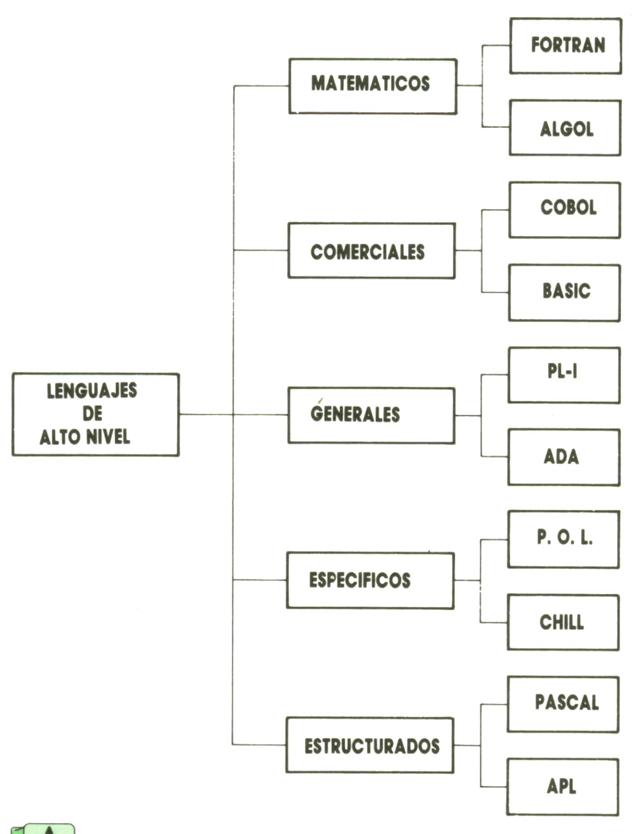


En todo cálculo de producen errores por diversas causas.

La precisión deseada, es decir, el máximo error admitido, repercute en el número de iteraciones a realizar, y por tanto, en el tiempo de proceso. El fin del proceso viene determinado por las características de los datos que se desean obtener, y por el error máximo admitido.

En cuanto a los ordenadores utilizados en proyectos de este tipo, no tienen por qué ser específicos, pero sí se caracterizan porque los periféricos que utilizan no son demasiado rápidos.

De la misma forma, no existen muchas limitaciones a la hora de elegir lenguaje de programación. De hecho se utilizan muchos lenguajes de alto nivel, aunque se consideran más convenientes el BASIC, FORTRAN, ALGOL y PASCAL.




Clasificación de los lenguajes de alto nivel.



Aplicaciones de gestión

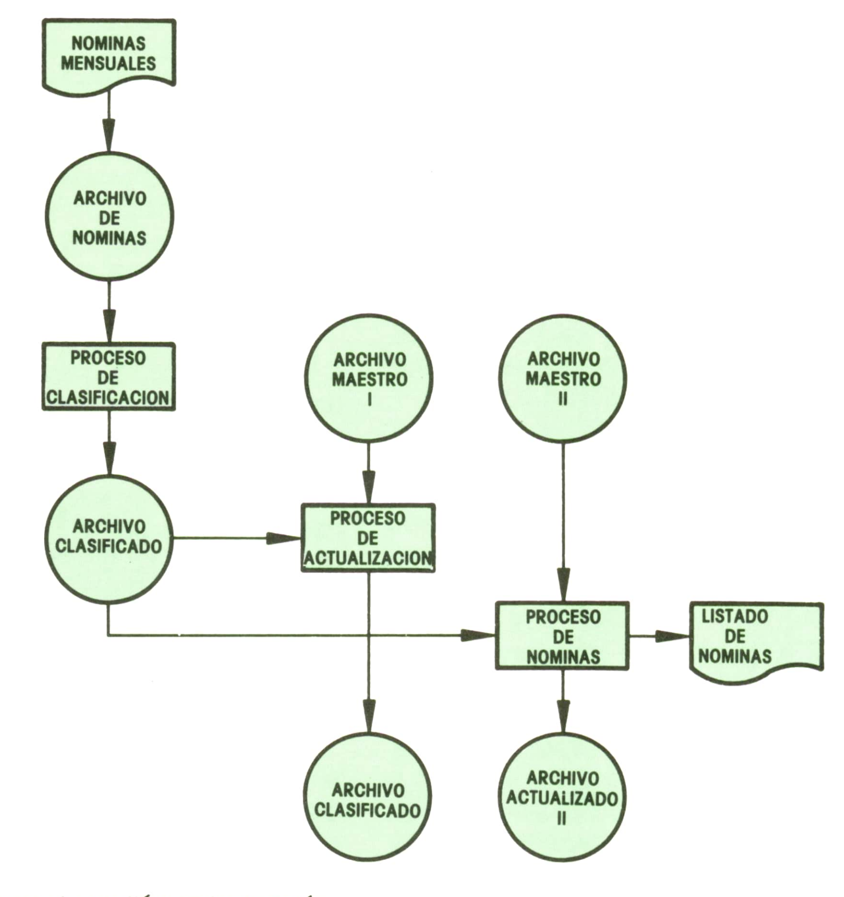
A diferencia de las aplicaciones técnicas, las de gestión manejan muchos más datos, siendo, sin embargo, las operaciones de tratamiento mucho más sencillas.

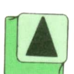
La gran cantidad de informaciones que se tratan obligan a tener que agrupar estos registros y ficheros, incluso a la utilización de bases de datos.

Las operaciones típicas que se realizan en aplicaciones de gestión son: actualizaciones, modificaciones y consultas. Operaciones que sólo afectan a registros, añadiendo o borrándolos de los ficheros. Este tipo de funciones no son aritméticas, pero no debe pensarse que no se van a utilizar.

También, a diferencia de la aplicación técnico-científica, las del tipo de gestión requieren un profundo estudio de las tres fases de análisis.

En la fase de análisis de viabilidad se aborda la elección de los problemas o aplicaciones a mecanizar y del ordenador más adecuado. Para ello se analiza




Ejemplo de programa de gestión: actualización.

la estructura de la empresa, el volumen de datos a tratar y el coste de la mecanización, y con los datos que se obtengan de este estudio se replantea el continuar con la mecanización.

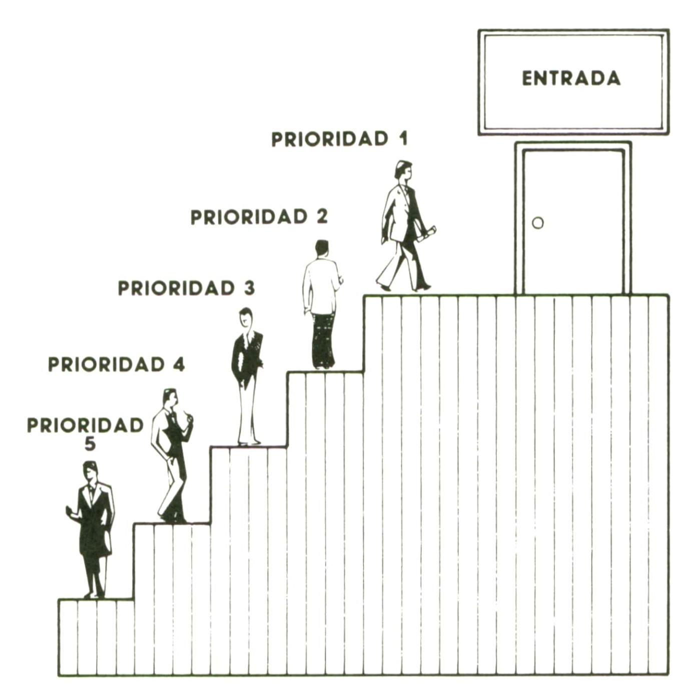
Basándose en los mismos resultados, se asignan prioridades a cada una de las tareas. Los criterios en los que se debe basar una asignación de prioridades son:

— Referentes al beneficio económico de la empresa. En el estudio previo de viabilidad se habían hecho cálculos del coste y beneficios de cada una de las áreas a mecanizar, bueno, pues éste es el punto en el que se decidirá cuáles de estas aplicaciones van a ser más rentables y, por tanto, van a tener más prioridad.

— Referentes al tiempo. Aplicaciones como nóminas suelen tener un tratamiento mensual, pero hay otro tipo de aplicaciones, como son las actualizaciones de ficheros históricos, en los que las variaciones no se producen tan a menudo, que tienen tratamientos más de tarde en tarde. De esta manera, se asigna mayor prioridad a trabajos que tengan que ejecutarse más frecuentemente.

— Referentes a la estructuración de los programas. Suelen tener prioridad aplicaciones estructuradas sobre las que no lo estén, al igual que es conveniente mecanizar primero los problemas más simples para pasar a los más complejos en una fase posterior.

Por otro lado, en las aplicaciones de gestión se necesita tener mucha y bien estructurada documentación. Hay que pensar que como resultado de las aplicaciones se obtendrán documentos e impresos de salida con destino a muchas

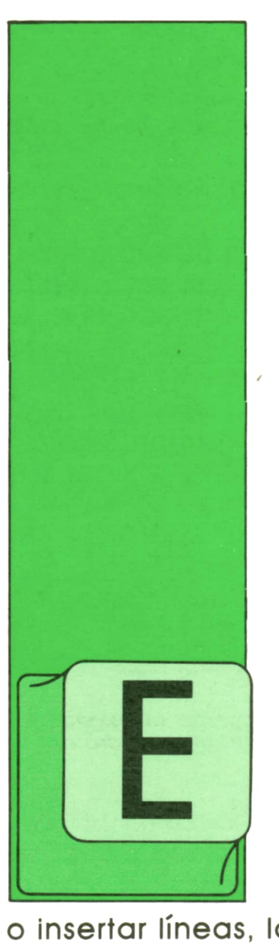



En las aplicaciones a mecanizar la asignación de prioridades indica qué trabajo tendrá entrada al sistema primero.

personas. El diseñar esta documentación es un trabajo bastante costoso para los analistas, teniendo en cuenta que el personal al que debe ser enviada esta documentación probablemente no será informático.

En cuanto a la selección del ordenador más adecuado se debe considerar la relación entre el coste de alquiler, en su caso, y la rentabilidad y ocupación del equipo. Además, es necesario prever el crecimiento de la aplicación y del sistema.

Los lenguajes más utilizados en este tipo de aplicaciones son el COBOL y el RPG.



MAQUINA 6502

Algo más sobre el lenguaje ensamblador

EL ASSEMBLER nos permite la entrada de programas en lenguaje máquina, al igual que lo hacemos con un programa BASIC. Podemos, por tanto, alterar, borrar o insertar líneas, las cuales van precedidas de su número de línea.

Veamos cuál debe ser el orden de entrada de datos en una línea:

1.°) Número de línea.

2.°) Marca (optativa). Una marca o etiqueta es una cadena de caracteres que se introduce para ejecutar los saltos sin necesidad de decir o calcular la distancia de salto ni la dirección de destino. Simplemente se coloca en la línea a la cual queremos que el procesador salte después de una instrucción BNE Marca, por ejemplo.

3.°) Comando ASSEMBLER y posibles operandos.

4.°) Comentario (opcional). Se introduce precedido de un punto y coma (;).

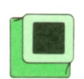
Al ensamblar el programa, el ensamblador calcula automáticamente las direcciones de todas las bifurcaciones, e inserta los códigos de los operandos y las direcciones asociadas en la memoria.

También pueden definirse al principio del programa las variables que vayamos a necesitar dándole un nombre y luego referirnos a ellas mediante dicho nombre y no mediante el número que contienen.

Por ejemplo, podemos definir BORDE = $DO20, y dentro del programa introducir STA BORDE e incluso STA BORDE + 1 (que sería el fondo).

Se trata de un ASSEMBLER de dos pasos, el primero que memoriza todas las etiquetas, definiciones de variables, etc., y el segundo que calcula las conversiones adecuadas para que el programa sea ejecutable.

Hay que tener en cuenta que no todos los monitores de código máquina permiten el uso de etiquetas y operaciones como las del ejemplo anterior. En este caso habrá que calcular el salto o la suma aparte y poner el valor final en la línea correspondiente.



Manejo de interrupciones

¡Por fin llegó el momento! Vamos a profundizar en las interrupciones del tipo IRQ, es decir, las evitables, y vamos a crear una rutina basada en ellas, la cual será ampliamente comentada, para que el lector pueda después poner en práctica sus propias ideas con esta técnica.

Lo más importante de las interrupciones del tipo IRQ es que se le puede «decir» al ordenador que deje de producirlas, momento que se aprovecha para añadir nuestra propia rutina o cambiar la existente. Después se vuelve a poner el señalizador de interrupciones, y ya está, ¡nuestra rutina se está ejecutando 60 veces por segundo, aunque no haya ningún programa en memoria!

Pero veamos cómo se hace todo esto. Es bastante fácil y el resultado puede ser espectacular.

El vector que apunta a la rutina de interrupt está situado en las posiciones $0314 (788) y $0315 (789).

Este vector suele contener los números 49 y 234, respectivamente. Mediante el cálculo que se explicó en capítulos anteriores se obtiene la dirección de inicio de la rutina:

Dirección = 49 + 256 · 234 = 59.953 ($EA31)

Pues bien, lo primero que tenemos que hacer es cambiar la dirección variando los valores del vector para que apunte donde nosotros queramos incorporar la nueva rutina. Pero antes debemos inhabilitar las interrupciones mediante el comando SEI.

Con ello hemos desplazado la rutina de interrupt a la posición $COOD en hexadecimal.

A continuación ponemos el señalizador de interrupciones CLI, y acabamos con RTS.

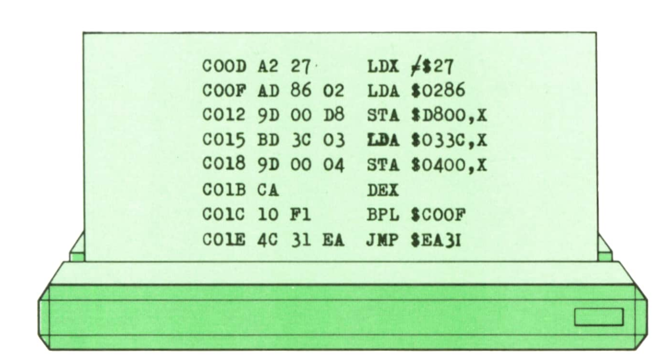
Tenemos, por tanto, el siguiente listado:

Ahora, a partir de la posición $COOD, empezamos la rutina que vamos a tratar.

Se trata de poner un letrero o encabezamiento en la primera línea de pantalla y que permanezca ahí aunque utilicemos la opción CRL-HOME de borrado.

Añada a las líneas que hemos explicado las siguientes y habrá terminado la rutina.

```
C000 78         SEI
C001 A9 0D      LDA #$0D
C003 8D 14 03   STA $0314
C006 A9 C0      LDA #$C0
C008 8D 15 03   STA $0315
C00B 58         CLI
C00C 60         RTS
```



```
C00D A2 27      LDX #$27
C00F AD 86 02   LDA $0286
C012 9D 00 D8   STA $D800,X
C015 BD 3C 03   LDA $033C,X
C018 9D 00 04   STA $0400,X
C01B CA         DEX
C01C 10 F1      BPL $C00F
C01E 4C 31 EA   JMP $EA31
```

Vamos a explicar cada línea de programa.

LDX $27: 40 caracteres en la línea.

LDA $0286: La posición $0286 = 646 controla el color de los caracteres en pantalla. Este «color» se almacena en el ACU.

STA $D800,X: La posición $D800 corresponde al color del primer carácter en la parte superior izquierda de la pantalla. Al ir variando X, se irá dando color a todos los caracteres que se encuentren en la primera línea de pantalla, según el valor que se encuentre en la posición $0286.

LDA $033C,x: Recoge cada uno de los caracteres que van a aparecer en la línea a partir de la posición $033C(828) cargándolo al ACU según varíe X.

STA $0400,X: Va colocando cada uno de los caracteres anteriores a partir de la posición $0400(1024), que es la primera correspondiente a la pantalla.

DEX: Decrementa el contenido del registro X en una unidad.

BPL $COOF: Hará que el programa bifurque mientras el flag N esté desactivado. Como el registro X va decrementándose desde $27(40), cuando llegue al valor «0» el flag N se activará y el programa sale del bucle.

JMP $EA31: Esta instrucción es fundamental y debe estar colocada al final de la rutina de interrupción como si fuese RTS. Lo que hace es continuar la rutina normal de interrupt.

Por supuesto, cada uno puede colocar los datos en la zona de memoria que mejor le parezca, sin más que cambiar la instrucción LDA $033C,X, por LDA $Dirección,X.

Por otra parte, en los programas anteriores he introducido el desensamblado completo, cosa que no hay que hacer al teclear el programa. Solamente debe teclearse la dirección de inicio ($COOO) como lo requiere su monitor, y a continuación los comandos y sus operandos (las dos columnas de la derecha).

Ahora ya sólo nos queda probar la rutina. Para ello, puede introducir un programa BASIC para escribir el mensaje.

Si no le gusta ese mensaje simplemente cámbielo por otro en la línea 70. Puede probar también otras variantes en la rutina de interrupciones.

Como puede comprobar, aunque borre la pantalla, o haga «NEW» y elimine

```
10  PRINT CHR$(147)
20  READ A$:A=A+1
30  IF A$="FIN" THEN SYS 49192:END
40  IF A$="" THEN POKE 827+A,32
50  IF A$   "" THEN POKE 827+A,ASC(A$)-64
60  GOTO 20
70  DATA ,,,,,,I,N,T,E,R,R,U,P,C,I,O,N,E,S,,
    I,N,T,E,R,C,E,P,T,A,D,A,S,,,,FIN
```

el programa en memoria, el letrero permanece ahí inalterable.

La única manera de desconectarlo sería pulsando las teclas RUN/STOP y RESTORE simultáneamente. Para volverlo a conectar, teclee SYS 49152 en modo directo.

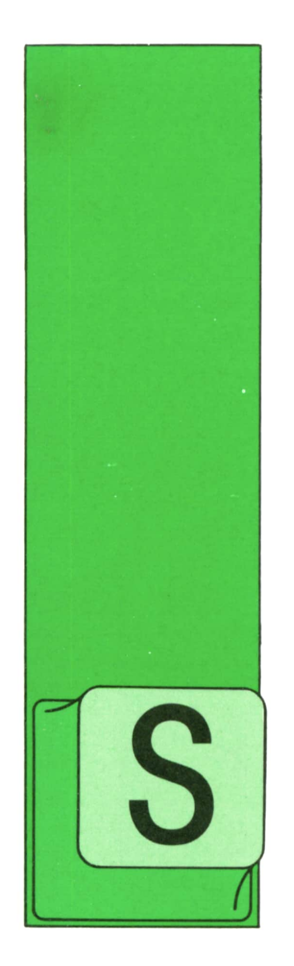
Si desactiva las teclas RUN/STOP y RESTORE con POKE 808,225:POKE 775,200 ya no podrá eliminar el letrero, a no ser que apague y vuelva a encender su COMMODORE-64.

Lo que no debe hacer es, una vez tecleado SYS 49152 ($COOO) y puesto el letrero en pantalla, volver a introducir el mismo comando SYS. No ocurre nada irreparable, pero la rutina puede fallar. Si es así, apague y vuelva a encender.

Pruebe a hacer todo lo que se le ocurra, siempre se aprende algo nuevo. ¡Que se divierta!

PROGRAMAS

EDUCATIVOS • DE UTILIDAD • DE GESTION • DE JUEGOS





Emisor-receptor de morse

SABER MORSE es algo que siempre nos puede venir bien. Nunca podemos saber cuándo lo vamos a necesitar y como su aprendizaje es muy sencillo, no cuesta nada intentarlo. Con este programa podremos aprender fácilmente a usarlo.

```
1000 REM ************************************************************************
1010 REM *                       EMISOR-LECTOR DE MORSE                         *
1020 REM ************************************************************************
1030 REM
1040 REM **************************************
1050 REM ******* (c) Ed. Siglo Cultural *******
1060 REM ******* (c) 1987               *******
1070 REM **************************************
1080 REM
1090 OPTION BASE 1
1100 DIM C$(28)
1110 CLS
1120 FOR F=1 TO 28
1130    READ C$(F)
1140 NEXT F
1150 SCREEN 0
1160 WIDTH 40
1170 REM
1180 REM ************************
1190 REM * CLAVES DE LAS LETRAS *
1200 REM ************************
1210 REM
1220 DATA "590000"
1230 DATA "955500"
1240 DATA "959500"
1250 DATA "955000"
1260 DATA "500000"
1270 DATA "559500"
1280 DATA "995000"
1290 DATA "555500"
1300 DATA "550000"
```

```
1310 DATA "599900"
1320 DATA "959000"
1330 DATA "595500"
1340 DATA "990000"
1350 DATA "950000"
1360 DATA "999000"
1370 DATA "599500"
1380 DATA "995900"
1390 DATA "595000"
1400 DATA "555000"
1410 DATA "900000"
1420 DATA "559000"
1430 DATA "555900"
1440 DATA "599000"
1450 DATA "955900"
1460 DATA "959900"
1470 DATA "995500"
1480 DATA "595959"
1490 DATA "559955"
1500 CLS
1510 LOCATE 3,6:PRINT "<<<< CODIFICADOR DE MORSE >>>>"
1520 LOCATE 7,5:PRINT "1-INTRODUCIR TEXTO"
1530 LOCATE 10,5:PRINT "2-INTRODUCIR CLAVES"
1540 LOCATE 13,5:PRINT "3-TEST DE TECLADO"
1550 LOCATE 20,5:PRINT " PULSE SU OPCION ==> ";
1560 LET K$=INKEY$
1570 IF K$="1" THEN PRINT K$:GOSUB 1620:GOTO 1500
1580 IF K$="2" THEN PRINT K$:GOSUB 1850:GOTO 1500
1590 IF K$="3" THEN PRINT K$:GOSUB 2360:GOTO 1500
1600 GOTO 1560
1610 REM
1620 REM ********************
1630 REM * INTRODUCIR TEXTO *
1640 REM ********************
1650 REM
1660 CLS
1670 INPUT "Introduce texto= ";A$
1680 FOR F=1 TO LEN(A$)
1690    LET B$=MID$(A$,F,1)
1700    LET N=ASC(B$)-64
1710    PRINT B$;
1720    IF B$=" " THEN FOR P=1 TO 15:NEXT P:GOTO 1810
1730    LET B$=C$(N)
1740    FOR C=1 TO 6
1750       LET G=VAL(MID$(B$,C,1))
1760       LET G=G*2
1770       LET E$=STR$(G)
1780       IF G<>0 THEN LET F$="O3 L"+E$+" C"
1790       PLAY F$
1800    NEXT C
1810 NEXT F
1820 GOSUB 2720
1830 RETURN
1840 REM
1850 REM *********************
1860 REM * INTRODUCIR CLAVES *
1870 REM *********************
1880 REM
1890 CLS
1900 PRINT "Introduce las claves con los signos '-' y '.' con un espacio entre l
etra y letra"
1910 PRINT "Para separar una palabra de otra pon 2 espacios.Al finalizar el text
o pon '+'."
1920 LET C=1
1930 FOR F=6 TO 18
1940    LET F$=""
1950    LET E$=""
1960    FOR R=1 TO 4
```

```
1970       IF MID$(C$(C),R,1)="9" THEN LET F$=F$+"-":ELSE IF MID$(C$(C),R,1)="5"
 THEN LET F$=F$+"."
1980       IF MID$(C$(C+1),R,1)="9" THEN LET E$=E$+"-":ELSE IF MID$(C$(C+1),R,1)
="5" THEN LET E$=E$+"."
1990    NEXT R
2000    LOCATE F,1:PRINT CHR$(C+64);CHR$(175);" ";F$,CHR$(C+65);CHR$(175);" ";E$
2010    LET C=C+2
2020 NEXT F
2030 PRINT ".";CHR$(175);" ";".-.-.-"
2040 PRINT "?";CHR$(175);" ";"..--.."
2050 PRINT:PRINT
2060 INPUT "INTRODUCE EL TEXTO: ";A$
2070 CLS
2080 LET S$=""
2090 FOR F=1 TO LEN(A$)
2100    LET S$=S$+MID$(A$,F,1)
2110    IF MID$(A$,F,1)=" " THEN GOSUB 2180:LET S$=""
2120    IF MID$(A$,F,2)="  " THEN PRINT " ";
2130    IF MID$(A$,F,1)="+" THEN GOSUB 2180:GOSUB 2720:RETURN
2140 NEXT F
2150 GOSUB 2840
2160 RETURN
2170 REM
2180 REM ****************
2190 REM * BUSCA CLAVES *
2200 REM ****************
2210 REM
2220 LET W$=""
2230 FOR C=1 TO LEN(S$)
2240    IF MID$(S$,C,1)="-" THEN LET W$=W$+"9"
2250    IF MID$(S$,C,1)="." THEN LET W$=W$+"5"
2260 NEXT C
2270 LET Q$="000000"
2280 LET W$=W$+MID$(Q$,1,6-LEN(W$))
2290 IF W$="559955" THEN PRINT "?";:RETURN
2300 IF W$="595959" THEN PRINT ".";:RETURN
2310 FOR C=1 TO 26
2320    IF W$=C$(C) THEN PRINT CHR$(64+C);
2330 NEXT C
2340 RETURN
2350 REM
2360 REM *******************
2370 REM * TEST DE TECLADO *
2380 REM *******************
2390 REM
2400 CLS
2410 PRINT "  Pulsa '-' para producir la raya (-)"
2420 PRINT "  Pulsa '.' para producir el punto (.)"
2430 PRINT "  Pulsa ENTER para imprimir letra"
2440 PRINT "  Pulsa ESPACE para poner espacios"
2450 PRINT "  Pulsa ESC para acabar"
2460 PRINT
2470 GOSUB 2840
2480 LOCATE 10,1
2490 PRINT "===> ";
2500 LET S$=""
2510 LET K$=INKEY$
2520 IF K$="-" THEN PLAY "O3 L20 C":LET S$=S$+"5"
2530 IF K$="." THEN PLAY "O3 L25 C":LET S$=S$+"9"
2540 IF K$=CHR$(27) THEN RETURN
2550 IF K$=CHR$(13) THEN GOSUB 2590:LET S$=""
2560 IF K$=" " THEN PRINT " ";
2570 GOTO 2510
2580 REM
2590 REM *************************
2600 REM * CODIFICAR PULSACIONES *
2610 REM *************************
2620 REM
```

```
2630 LET Q$="000000"
2640 LET S$=S$+MID$(Q$,1,6-LEN(S$))
2650 IF S$="595959" THEN PRINT ".";:RETURN
2660 IF S$="559955" THEN PRINT "?";:RETURN
2670 FOR F=1 TO 26
2680     IF C$(F)=S$ THEN PRINT CHR$(64+F);
2690 NEXT F
2700 RETURN
2710 REM
2720 REM **********************
2730 REM * FINAL DE LA OPCION *
2740 REM **********************
2750 REM
2760 LOCATE 21,25
2770 PRINT "PULSA UNA TECLA PARA TERMINAR."
2780 LET K$=INKEY$
2790 IF K$="" THEN GOTO 2780
2800 LOCATE 21,25
2810 PRINT SPACE$(40)
2820 RETURN
2830 REM
2840 REM **********************
2850 REM * CONTINUAR LA OPCION *
2860 REM **********************
2870 REM
2880 LOCATE 21,25
2890 PRINT "PULSA UNA TECLA PARA CONTINUAR."
2900 LET K$=INKEY$
2910 IF K$="" THEN GOTO 2900
2920 LOCATE 21,25
2930 PRINT SPACE$(40)
2940 RETURN
```

El programa fue realizado en un IBM pc bajo GWBASIC. Por ello, cualquier ordenador que disponga de este dialecto del BASIC puede ejecutar este programa sin cambios. Para el resto de los ordenadores, se proponen los siguientes:

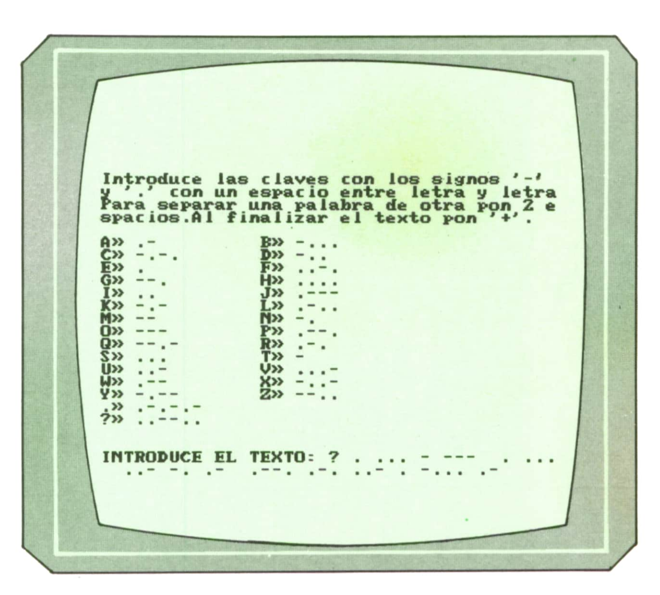
AMSTRAD:

```
1090 REM
1150 MODE
1160 REM
1510 LOCATE 6,3: PRINT
     "<<<<CODIFICADOR DE
     MORSE>>>>"
1520 LOCATE 7,5: PRINT "1-INTRODUCIR
     TEXTO"
1530 LOCATE 10,5: PRINT "2-INTRODUCIR
     CLAVES"
1540 LOCATE 13,5: PRINT "3-TEST DE
     TECLADO"
1550 LOCATE 20,5: PRINT "PULSE SU
     OPCION → ";
1790 FOR I = 1 TO G: PRINT CHR$(7);
     NEXT I
2000 LOCATE 1,F: PRINT CHR$ (C + 64);
     CHR$(175); " "; F$,CHER$(C + 65
     CHR$(175); " "; E$
2480 LOCATE 1,10
2520 IF K$ = "—" THEN PRINT CHR$(7):
     LET S$ = S$ + "5"
2530 IF K$ = "." THENPRINT CHR$(7);
     CHR$(7): LET S$ = S$ + "9".
2760 LOCATE 25,21
2800 LOCATE 25,21
2880 LOCATE 25,21
2920 LOCATE 25,21
```

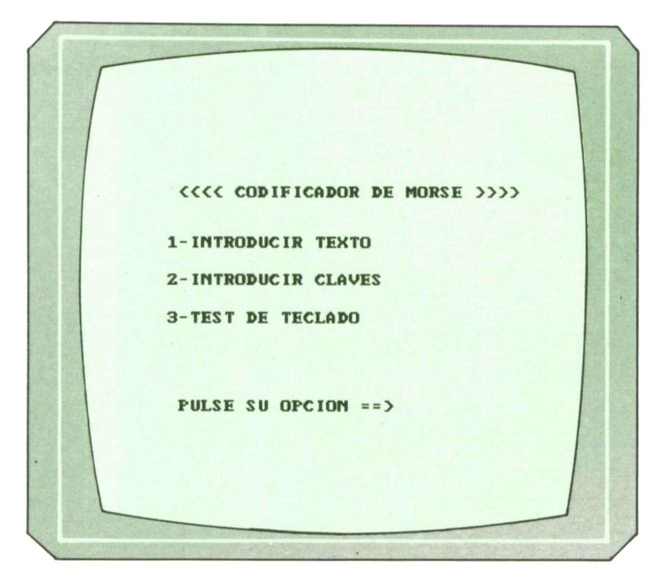
MSX:

```
1510 LOCATE 6,3: PRINT
     "<<<<CODIFICADOR DE
     MORSE>>>>"
1520 LOCATE 7,5: PRINT "1-INTRODUCI
     TEXTO"
```

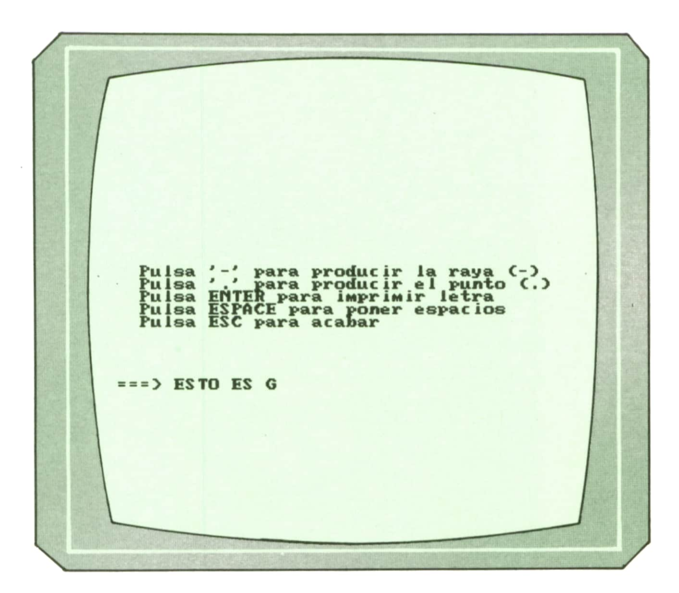
```
1530 LOCATE 10,5: PRINT "2-INTRODUCIR
     CLAVES"
1540 LOCATE 13,5: PRINT "3-TEST DE
     TECLADO"
1550 LOCATE 20,5: PRINT "PULSE SU
     OPCION → ";
1790 FOR I = 1 TO G: PRINT CHR$(7);
     NEXT I
2000 LOCATE 1,F: PRINT CHR$(C + 64);
     CHR$(175); " "; F$, CHR$ (C + 65);
     CHR$(175); " "; E$.
2480 LOCATE 1,10
2520 IF K$ = "—" THEN PRINT CHR$(7):
     LET S$ = S$ + "5"
2530 IF K$ = "." THEN PRINT CHR$(7);
     CHR$(7): LET S$ = S$ + "9"
2760 LOCATE 25,21
2800 LOCATE 25,21
2880 LOCATE 25,21
2920 LOCATE 25,21
```




El test del teclado.




Menú de opciones.



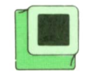

Introducción de código Morse.

El programa nos permite tres funciones distintas:

1. *Introducir texto.* Una vez introducido el texto, el ordenador nos lo escribirá como rayas y puntos y nos mostrará cómo suena.

2. *Introducir claves.* Nosotros introducimos el código Morse y el ordenador lo interpreta y nos muestra el texto que hemos escrito.

3. *Test de teclado.* Lo utilizaremos para familiarizarnos con el teclado y para coger velocidad.



Scroll de atributos para Spectrum

Con este programa en código máquina podremos hacer SCROLL de la zona de atributos (colores) del SPECTRUM en cualquiera de las cuatro direcciones. El primer programa que vamos a ver, aparte de introducir el código máquina necesario para que el programa funcione, nos hace una demostración.

```
  10 REM ***********************
  20 REM *                     *
  30 REM * SCROLL DE ATRIBUTOS *
  40 REM *                     *
  50 REM *    PROGRAMA DEMO    *
  60 REM *    -------------    *
  70 REM *                     *
  80 REM ***********************
  90 REM
 100 REM ********************************
 110 REM *                              *
 120 REM * (c) Ediciones Siglo Cultural *
 130 REM *                              *
 140 REM * (c) 1987                     *
 150 REM *                              *
 160 REM ********************************
 170 REM
1000 PAPER 0
1010 BORDER 0
1020 INK 6
1030 CLEAR 54999
1060 PRINT AT 10,10; FLASH 1;"CARGANDO DATAS"
1070 PRINT AT 12,8; FLASH 1; INVERSE 1; INK 4;"ESPERE UN MOMENTO."
1080 REM
1090 REM ******************************
1100 REM * LECTURA DEL CODIGO MAQUINA *
1110 REM ******************************
1120 REM
1130 LET TOT=0
1140 LET LIN=9040
1150 FOR I=55000 TO 55164 STEP 15
1160    LET CHEK=0
1170    LET LIN=LIN+10
1180    FOR J=I TO I+14
1190       READ A
1200       LET CHEK=CHEK+A
1210       POKE J,A
1220    NEXT J
1230    READ TOT
1240    IF TOT<>CHEK THEN  GO TO 5000
1250 NEXT I
2000 REM
2010 REM ****************
2020 REM *              *
2030 REM * DEMOSTRACION *
2040 REM *              *
2050 REM ****************
2060 REM
2070 CLS
2080 OVER 1
2090 FOR I=0 TO 7
2100    PAPER I
2110    FOR J=0 TO 21
2120       PRINT AT J,I*3+2;"   "
2130    NEXT J
2140 NEXT I
2150 OVER 0
2160 PAUSE 50
2170 REM
2180 REM *************************
2190 REM * SCROLL A LA IZQUIERDA *
2200 REM *************************
2210 REM
2220 POKE 23300,1 :  REM SCROLL A LA IZQUIERDA
2230 POKE 23301,7 :  REM NUMERO DE LINEAS. DESDE LA 0 A LA 6
2240 POKE 23302,7 :  REM COLOR QUE APARECE EN LA PARTE DERECHA
2250 REM
```

```
2260 FOR I=0 TO 31
2270    RANDOMIZE USR 55000
2280    PAUSE 4
2290 NEXT I
2295 PAUSE 50
2300 REM
2310 REM ***********************
2320 REM * SCROLL A LA DERECHA *
2330 REM ***********************
2340 REM
2350 POKE 23300,2 :  REM SCROLL A LA DERECHA
2360 POKE 23301,12:  REM NUMERO DE LINEAS. DESDE LA 0 A LA 9
2370 POKE 23302,7 : REM COLOR QUE APARECE EN LA PARTE IZQUIERDA
2380 REM
2390 FOR I=0 TO 31
2400    RANDOMIZE USR 55000
2410    PAUSE 4
2420 NEXT I
2430 PAUSE 50
2440 REM
2450 REM ***********************
2460 REM * SCROLL HACIA ARRIBA *
2470 REM ***********************
2480 REM
2490 POKE 23300,3 :  REM SCROLL HACIA ARRIBA
2500 POKE 23301,21: REM NUMERO DE LINEAS. DESDE LA 0 A LA 19
2510 POKE 23302,7 : REM COLOR QUE APARECE EN LA PARTE INFERIOR
2520 REM
2530 FOR I=0 TO 10
2540    RANDOMIZE USR 55000
2550    PAUSE 4
2560 NEXT I
2570 PAUSE 50
2580 REM
2590 REM **********************
2600 REM * SCROLL HACIA ABAJO *
2610 REM **********************
2620 REM
2630 POKE 23300,4 : REM SCROLL HACIA ABAJO
2640 POKE 23301,21: REM NUMERO DE LINEAS. DESDE LA 0 A LA 19
2650 POKE 23302,7 : REM COLOR QUE APARECE EN LA PARTE SUPERIOR
2660 REM
2670 FOR I=0 TO 10
2680    RANDOMIZE USR 55000
2690    PAUSE 4
2700 NEXT I
2710 PAUSE 50
2720 REM
2730 REM *** REPETICION **
2735 PAPER 0
2740 REM
2750 CLS
2760 FOR I=0 TO 20
2770    FOR J=0 TO I
2780       PRINT " ";
2790    NEXT J
2800    PRINT "SCROLL"
2810  NEXT I
2820 PRINT AT 0,0;
2830 GO TO 2080
4999 PAPER 0: STOP
5000 REM
5010 REM ****************************
5020 REM * ERROR EN LAS LINEAS DATA *
5030 REM ****************************
5040 REM
5050 CLS
5060 PRINT "HA HABIDO UN ERROR EN LAS DATA"
```

```
5070 PRINT : PRINT
5080 PRINT "EL ERROR ESTA EN LA LINEA ";LIN
5090 PRINT : PRINT
5100 PRINT "REPASA DICHA LINEA"
5110 PRINT : PRINT
5120 LIST LIN
5130 STOP
9000 REM
9010 REM ************************************
9020 REM * LINEAS DATA CON EL CODIGO MAQUINA *
9030 REM ************************************
9040 REM
9050 DATA 0,58,4,91,254,1,40,13,254,2,40,34,254,3,40
9055 DATA 1088
9060 DATA 62,254,4,40,91,201,33,0,88,58,5,91,79,58,6
9065 DATA 1070
9070 DATA 91,6,31,35,94,43,115,35,16,249,119,35,13,32,242
9075 DATA 1156
9080 DATA 201,33,255,87,17,32,0,58,5,91,71,79,25,16,253
9085 DATA 1223
9090 DATA 58,6,91,6,31,43,94,35,115,43,16,249,119,43,13
9095 DATA 962
9100 DATA 32,242,201,33,0,0,17,32,0,58,5,91,71,25,16
9105 DATA 823
9110 DATA 253,68,77,33,32,88,17,0,88,237,176,58,6,91,6
9115 DATA 1230
9120 DATA 32,18,19,16,252,201,33,0,0,17,32,0,58,5,91
9125 DATA 774
9130 DATA 71,25,16,253,68,77,197,33,0,88,71,25,16,253,229
9135 DATA 1422
9140 DATA 25,235,225,193,237,184,58,6,91,6,32,17,0,88,18
9145 DATA 1415
9150 DATA 19,16,252,201,0,0,0,0,0,0,0,0,0,0,0
9155 DATA 488
```

Hay que tener mucho cuidado al introducir las líneas DATA para no equivocarnos. De todas maneras, si lo hacemos no importa, ya que el programa nos dirá en qué línea lo hemos hecho.

Para ejecutar esta rutina hay que pokear una serie de valores en memoria. Estos son:

1. En la posición 23300 pokearemos la dirección en que queremos que se produzca el SCROLL según la siguiente tabla:

1 = SCROLL a la izquierda
2 = SCROLL a la derecha
3 = SCROLL hacia arriba
4 = SCROLL hacia abajo

2. En la posición 23301 pokearemos el número de líneas que queremos SCROLLear. Dicho número puede estar comprendido entre 1 y 24.

3. En la posición 23302 introduciremos el atributo que tendrá la línea que queda sin atributos tras el SCROLL. El atributo es un número entre 0 y 255 que nos indica el color del papel, de la tinta, el brillo y el flash de la siguiente manera:

— Los primeros tres BITs me dicen la tinta.

— Los BITs 3, 4 y 5 (los tres siguientes) me dicen el papel.

— El siguiente BIT (el 6) me indica si es con brillo (1) o sin él (0).

— El último BIT informa de si hay FLASH (1) o no (0).

A continuación incluimos el programa fuente para todos aquellos que les interese ver cómo está hecho el programa.

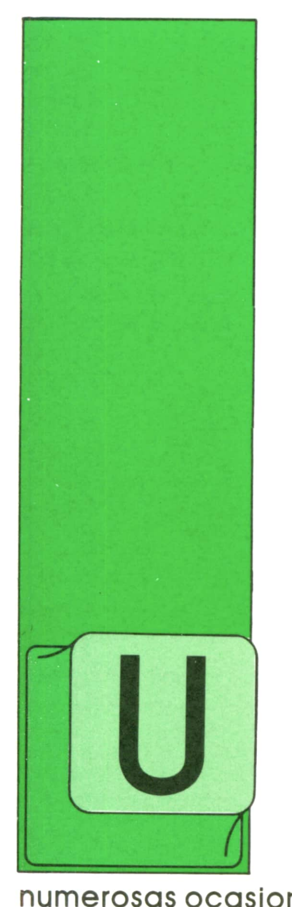
```
1000 ;******************************************
1010 ;*                                        *
1020 ;*   R U T I N A S   D E   S C R O L L   *
1030 ;*    D E   L O S   A T R I B U T O S    *
1040 ;*                                        *
1050 ;******************************************
1060 ;
1070 ;
1080 ;*********************************
1090 ;*                               *
1100 ;* (C) EDICIONES SIGLO CULTURAL *
1110 ;*                               *
1120 ;* (C) 1987                      *
1130 ;*                               *
1140 ;*********************************
1150 ;
1160 ;
1170 ;=============================================
1180 ;=                                           =
1190 ;=  COMO EL PROGRAMA  PUEDE REALIZAR SCROLL =
1200 ;= EN LAS CUATRO DIRECCIONES DE LA PANTALLA =
1210 ;= TENEMOS QUE  ALMACENAR EN LA POSICION DE =
1220 ;= MEMORIA 23301  LA DIRECCION QUE QUEREMOS =
1230 ;= QUE TENGA  EL SCROLL  SEGUN LA TABLA QUE =
1240 ;= APARECE A CONTINUACION:                  =
1250 ;=                                           =
1260 ;=         1 = SCROLL A LA IZQUIERDA        =
1270 ;=         2 = SCROLL A LA DERECHA          =
1280 ;=         3 = SCROLL HACIA ARRIBA          =
1290 ;=         4 = SCROLL HACIA ABAJO           =
1300 ;=                                           =
1310 ;=============================================
1320 ;
1330 ;
1340 DIR         EQU     23300
1350 ;
1360 ;
1370 ;=============================================
1380 ;=                                           =
1390 ;=  EL NUMERO DE LINEAS QUE QUEREMOS MOVER  =
1400 ;= CADA VEZ QUE SE EJECUTE LA RUTINA TENE-  =
1410 ;= MOS QUE ALMACENARLAS EN LA DIRECCION DE  =
1420 ;= MEMORIA 23301.                            =
1430 ;=  DICHO NUMERO ESTARA  COMPRENDIDO ENTRE  =
1440 ;= 1 Y 24 AMBOS INCLUSIVE.                   =
1450 ;=                                           =
1460 ;=============================================
1470 ;
1480 ;
1490 NUM         EQU     23301
1500 ;
1510 ;
1520 ;============================================
1530 ;=                                          =
1540 ;=  EL ATRIBUTO QUE TENDRA LA'LINEA QUE SE =
1550 ;= QUEDA VACIA TRAS EL SCROLL A DE IR ALMA =
1560 ;= CENADO EN LA DIRECCION 233302.          =
1570 ;=  EL ATRIBUTO  SERA UN  NUMERO ENTRE 0 Y =
1580 ;= 255!                                     =
1590 ;=                                          =
1600 ;============================================
1610 ;
1620 ;
1630 ATTR        EQU     23302
```

```
1640 ;
1650 ;
1660 ;=============================================
1670 ;=                                           =
1680 ;=  ESTE PROGRAMA ES TOTALMENTE REUBICABLE,  =
1690 ;= POR ELLO,  AUNQUE EL PROGRAMA SE REALIZO  =
1700 ;= A PARTIR DE LA DIRECCION 55000, EL USUA-  =
1710 ;= RIO PUEDE ELEGIR LA DIRECCION QUE MAS LE  =
1720 ;= CONVENGA PARA SU PROGRAMA.                =
1730 ;=                                           =
1740 ;=============================================
1750 ;
1760 ;
1770            ORG     55000
1780 ;
1790 ;
1800 ;*********************************
1810 ;*********************************
1820 ;****** PROGRAMA PRINCIPAL ******
1830 ;*********************************
1840 ;*********************************
1850 ;
1860 ;
1870            LD      A,(DIR)         ; ALMACENAMOS LA DIRECCION DEL SCROLL
1880 ;
1890            CP      1
1900            JR      Z,IZQ           ; SI ES 1 VAMOS A 'IZQ'
1910 ;
1920            CP      2
1930            JR      Z,DER           ; SI ES 2 VAMOS A 'DER'
1940 ;
1950            CP      3
1960            JR      Z,ARR           ; SI ES 3 VAMOS A 'ARR'
1970 ;
1980            CP      4
1990            JR      Z,ABB           ; SI ES 4 VAMOS A 'ABB'
2000 ;
2010            RET                     ; SI ES OTRO NUMERO VOLVEMOS AL BASIC
2020 ;
2030 ;
2040 ;**************************
2050 ;*                        *
2060 ;*  SCROLL A LA IZQUIERDA  *
2070 ;*                        *
2080 ;**************************
2090 ;
2100 ;
2110 IZQ        LD      HL,22528        ; HL = AREA DE ATRIBUTOS
2120            LD      A,(NUM)         ;  A = NUMERO DE LINEAS A MOVER
2130            LD      C,A             ;  C = A
2140            LD      A,(ATTR)        ;  A = COLOR DE LA LINEA VACIA
2150 ;
2160 LIN-IZ     LD      B,31            ;  B = NUMERO DE CHR POR LINEA
2170 ;
2180 CHR-IZ     INC     HL              ; HL = HL + 1
2190            LD      E,(HL)          ;  E = CONTENIDO DE HL
2200            DEC     HL              ; HL = HL - 1
2210            LD      (HL),D          ; HL = CONTENIDO DE LA POSICION ANT.
2220            INC     HL              ; HL = HL + 1
2230            DJNZ    CHR-IZ          ; SI NO HEMOS TERMINADO CONTINUAMOS
2240 ;
2250            LD      (HL),A          ; PONEMOS EL ATRIBUTO
2260            INC     HL              ; HL = HL + 1
```

```
2270            DEC     C               ; UNA LINEA MENOS. C = C - 1
2280            JR      NZ,LIN-IZ       ; CONTINUAMOS SI QUEDAN LINEAS
2290 ;
2300            RET                     ; Y SI NO VOLVEMOS AL BASIC.
2310 ;
2320 ;
2330 ;*************************
2340 ;*                       *
2350 ;*  SCROLL A LA DERECHA  *
2360 ;*                       *
2370 ;*************************
2380 ;
2390 ;
2400 DER        LD      HL,22527        ; HL = ZONA DE ATRIBUTOS - 1
2410            LD      DE,32           ; DE = NUMERO DE BYTES POR LINEA
2420            LD      A,(NUM)         ;  A = NUMERO DE LINEAS
2430            LD      B,A             ;  B = A
2440            LD      C,A             ;  C = A
2450 ;
2460 SUM-DE     ADD     HL,DE           ; HL = HL + DE
2470            DJNJ    SUM-DE          ;  B = B - 1! IF B<>0 THEN SUM-DE
2480                                    ;  B = DIRECCION DE LA ULTIMA LINEA
2490 ;
2500            LD      A,(ATTR)        ;  A = ATRIBUTO A INTRODUCIR
2510 ;
2520 LIN-DE     LD      B,31            ;  B = NUMERO DE CHR POR LINEA - 1
2530 ;
2540 CHR-DE     DEC     HL              ; HL = HL - 1
2550            LD      E,(HL)          ;  E = LO QUE HAY EN HL
2560            INC     HL              ; HL = HL + 1
2570            LD      (HL),E          ; GUARDAMOS LO QUE HABIA EN E
2580            DEC     HL              ; HL = HL - 1
2590            DJNZ    CHR-DE          ; B = B - 1:IF B<>0 THEN CHR-DE
2600 ;
2610            LD      (HL),A          ; PONEMOS EL ATRIBUTO
2620            DEC     HL              ; HL = HL - 1
2630            DEC     C               ;  C = C - 1
2640            JR      NZ,LIN-DE       ; IF C<>0 THEN LIN-DE
2650 ;
2660            RET                     ; VOLVEMOS AL BASIC.
2670 ;
2680 ;
2690 *;*************************
2700 ;*                       *
2710 ;*  SCROLL HACIA ARRIBA  *
2720 ;*                       *
2730 ;*************************
2740 ;
2750 ;
2760 ARR        LD      HL,0            ; HL = 0
2770            LD      DE,32           ; DE = NUMERO DE CHR POR LINEA
2780            LD      A,(NUM)         ;  A = NUMERO DE LINEAS A MOVER
2790            LD      B,A
2800 ;
2810 SUM-AR     ADD     HL,DE           ; HL = HL + DE
2820            DJNZ    SUM-AR          ;  B = B - 1:IF B<>0 THEN SUM-AR
2830                                    ; HL = NUMERO D  BYTES A MOVER
2840 ;
2850            LD      B,H
2860            LD      C,L             ; BC = HL. BC = NUMERO DE BYTES A MOVER
2870            LD      HL,22560        ; HL = ORIGEN PARA LDIR
2880            LD      DE,22528        ; DE = DESTINO PARA LDIR
2890            LDIR                    ; LD (DE),(HL):HL=HL+1:DE=DE+1:BC=BC-1
2900 ;
```

```
2910           LD      A,(ATTR)          ;  A = ATRIBUTO A INTRODUCIR
2920           LD      B,32              ;  B = CARACTERES POR LINEA
2930 ;
2940 CHR-AR    LD      (DE),A            ; PONEMOS EL ATRIBUTO
2950           INC     DE                ; DE = DE + 1
2960           DJNZ    CHR-AR            ;  B = B - 1:IF B<>0 THEN CHR-ARR
2970 ;
2980           RET                       ; VUELTA AL BASIC
2990 ;
3000 ;
3010 ;************************
3020 ;*                      *
3030 ;*  SCROLL HACIA ABAJO  *
3040 ;*                      *
3050 ;************************
3060 ;
3070 ;
3080 ABB       LD      HL,0              ; HL = 0
3090           LD      DE,32             ; DE = NUMERO DE CHR POR LINEA
3100           LD      A,(LIN)           ;  A = NUMERO DE LINEAS A MOVER
3110           LD      B,A               ;  B = A
3120 ;
3130 SU1-AB    ADD     HL,DE             ; HL = HL + DE
3140           DJNZ    SU1-AB            ;  B = B - 1:IF B<>0 THEN SU1-AB
3150                                     ; HL = NUMERO DE BYTES A MOVER
3160 ;
3170           LD      B,H               ;  B = H
3180           LD      C,L               ;  C = L
3190                                     ; BC = NUMERO DE BYTES A MOVER
3200           PUSH    BC                ; GUARDAMOS BC EN EL STACK
3210 ;
3220           LD      HL,22528          ; HL = PRIMERA DIRECCION DE LOS ATTRIB
3230           LD      B,A               ;  B = NUMERO DE LINEAS A MOVER
3240 ;
3250 SU2-AB    ADD     HL,DE             ; HL = HL + DE
3260           DJNZ    SU2-AB            ;  B = B - 1:IF B<>0 THEN SU2-AB
3270                                     ; HL = DIRECCION DE LA SEGUNDA LINEA
3280 ;
3290           PUSH    HL                ; GUARDAMOS HL EN EL STACK
3300           ADD     HL,DE             ; HL = HL + DE
3310           EX      DE,HL             ; SWAP HL,DE
3320                                     ; DE = DIRECCION DE LA PRIMERA LINEA
3330           POP     HL                ; HL = DIRECCION DE LA SEGUNDA LINEA
3340           POP     BC                ; BC = NUMERO DE BYTES A MOVER
3350           LDDR                      ; LD (DE),(HL):DE=DE-1:HL=HL-1:BC=BC-1
3360 ;
3370           LD      A,(ATTR)          ;  A = ATRIBUTO A INTRODUCIR
3380           LD      B,32              ;  B = NUMERO DE CHR POR LINEA
3390           LD      DE,22528          ; DE = DIRECCION DONDE PONER ATRIBUTOS
3400 ;
3410 CHR-AB    LD      (DE),A            ; PONEMOS EL ATRIBUTO
3420           INC     DE                ; DE = DE + 1
3430           DJNZ    CHR-AB            ;  B = B - 1:IF B<>0 THEN CHR-AB
3440 ;
3450           RET                       ; VUELTA AL BASIC
```

TECNICAS DE ANALISIS





Comprobación de programas

UNO de los aspectos más delicados de la puesta en marcha de un sistema informático es la prueba de los programas y subsistemas que lo forman. En efecto, en numerosas ocasiones las tareas a realizar por una unidad de tratamiento son relativamente sencillas y no es complicado comprobar que dicho programa procesa los datos como está previsto; sin embargo, hay ocasiones en que los procesos son complejos, los resultados difícilmente previsibles o el programa de que se trata trabaja con los resultados de algún proceso anterior: en este tipo de situaciones el establecer las pruebas a que hay que someter al programa (y la comprobación de los resultados obtenidos) puede ser sumamente complejo y/o tedioso.

A pesar de ello, es muy importante, en la mayoría de los casos, el realizar los tests y pruebas necesarios hasta poder asegurar que el funcionamiento del programa se adecúa a lo que se espera de él.

Por otro lado, además de la comprobación de la «corrección» del proceso es muy conveniente someter los programas a otros controles, habrá que asegurarse de la «fiabilidad» del sistema (o de la parte de él que estemos considerando en ese momento), habrá que tomar cuenta y estudiar las «prestaciones» de cada módulo, habrá, en fin, que verificar la eficiencia de los circuitos externos de flujo de la información.

Al aludir a la fiabilidad de un sistema o subsistema, nos referimos a la cualidad de control de las actividades que realiza. Por un lado, deben existir suficientes procedimientos de comprobación de la inexistencia de errores: bien de errores que puedan introducirse desde el exterior, en los datos que se suministran al sistema (deben comprobarse los márgenes dentro de los que han de estar los diferentes datos de entrada, el tipo de datos aportados, el formato de ellos, etc.), bien de errores que introduzca el propio proceso debidos al tipo inadecuado (erróneo) de datos aportados.

Por otro lado, deben controlarse de un modo automático las operaciones y manipulaciones de los diferentes procesos y/o archivos que se realizan manualmente desde el exterior del sistema: pueden aparecer resultados erróneos (y muchas veces impredecibles) si se pone como fichero de entrada a un proceso, un archivo que no es el adecuado (un fichero distinto o una versión inadecuada del fichero correcto) o si se ejecuta, detrás de un programa determinado, una parte incorrecta del proceso, que no es la que corresponde.

Es muy importante también el examen de las prestaciones de un sistema, aunque usualmente no es un problema que se aborde seria y concienzudamente en el diseño y prueba de los sistemas (normalmente, en este tema, es de aplicación el dicho de que «no nos acordamos de Santa Bárbara hasta que truena»).

Entendemos aquí por «prestaciones de

COMPROBACION DE PROGRAMAS

A. CORRECCION DEL PROCESO

A.1. PREPARACION DE LOS DATOS

- ENTRADA DE DATOS.
- CONTENIDO DEL JUEGO DE ENSAYO (muestra o juego completo, casos típicos y atípicos, casos aleatorios, volumen de datos, etc.).
- GENERACION DE LOS DATOS (a partir de los datos de entrada, de las tablas de proceso o por generación automática).

A.2. COMPROBACION DE RESULTADOS

B. CONTROLES COMPLEMENTARIOS

B.1. FIABILIDAD DEL PROCESO

- CONTROL DE ERRORES (de datos externos o internos).
- CONTROL DE OPERACIONES MANUALES (ficheros inadecuados, secuencia incorrecta de proceso, etcétera).

B.2. PRESTACIONES DE LOS DISTINTOS MODULOS

- TIEMPOS DE PROCESO
  - INDIVIDUALES
  - EN LOS DIFERENTES ENTORNOS POSIBLES.
- TIEMPO DE MANIPULACIÓN

B.3. EFICIENCIA DE LOS CIRCUITOS DE INFORMACION.

- ELIMINACIÓN DE TIEMPOS MUERTOS O COMPLEJIDADES INNECESARIAS.

un sistema» la rapidez con que ejecuta los procesos para los que ha sido diseñado. El tiempo total que consume un conjunto de programas en ejecutarse es el resultado de los tiempos de proceso de cada programa más los tiempos de manipulación externa de las distintas operaciones a realizar entre procesos.

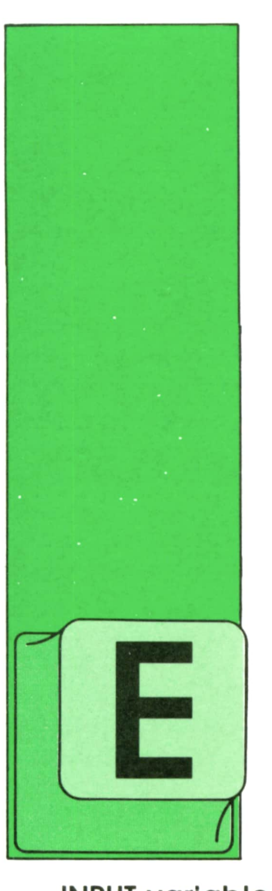
Este tiempo de manipulación entre cada programa depende enormemente del operador que los realice, del interés que ponga en la tarea... y hasta de su estado de ánimo: normalmente se hace una estimación aproximada y promedio, aunque en situaciones en que este tiempo es crítico (por el volumen de las operaciones a realizar o por el poco tiempo disponible) puede ser útil llegar a medir tiempos y establecer un cronograma concreto y preciso que se impone al operador.

La medida del tiempo de proceso consumido por cada programa no debe comportar, por el contrario, demasiadas dificultades, normalmente; sin embargo, en entornos complejos de trabajo (por ejemplo, cuando existen numerosos terminales trabajando simultáneamente, que producen una carga de trabajo muy variable dependiendo de la hora del día, o cuando los procesos que consumen mucho tiempo de CPU pueden coincidir o no con el que estamos estudiando), el estudio puede ser más difícil.

Naturalmente, el tiempo propio de proceso del programa no varía con estas circunstancias (sino que depende de la cantidad y tipo de los datos que ha de procesar), pero el tiempo de reloj consumido desde que comienza el proceso hasta que se pueda dar por concluido, sí puede ser muy diverso.

Por esta razón, deben establecerse dos tipos de pruebas: unas para evaluar el tiempo propio del proceso del programa y otras para ver la incidencia de los restantes procesos que puedan concurrir con el que nos interesa, en un momento dado. Normalmente, pueden ser extrapolables los tiempos de proceso obtenidos con una muestra, para determinar el tiempo que se consumirá cuando se realice el proceso real: esto simplifica no sólo la prueba en sí, sino (lo que es más importante) el trabajo de preparación de los datos de ensayo; pero es muy importante cerciorarse de que el volumen de datos no hace crecer el tiempo de modo no lineal (a veces, incluso, de forma exponencial), como sucede con la mayoría de los procesos de clasificación o de comprobación de «todos contra todos», porque en una situación de este tipo es básico realizar la prueba con un volumen de datos que sea representativo, para poder tener la certeza de que es válida la estimación obtenida.

Por último, es necesario evaluar, asimismo, la eficiencia de los circuitos externos de manejo de la información que el sistema supone, para confirmar que no están introduciendo complejidades o tiempos muertos innecesarios, que invaliden o minimicen las ventajas que el sistema aporta.



TECNICAS DE PROGRAMACION

ENTRADA Y SALIDA POR LA CONSOLA

(CONTINUACION)

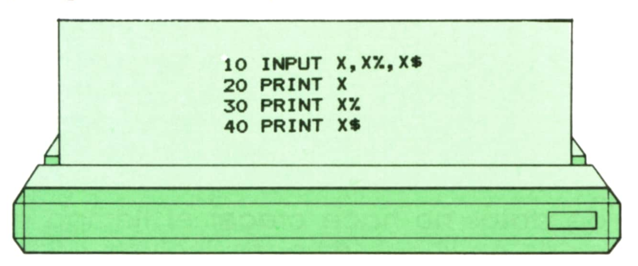


Otras formas de la instrucción INPUT

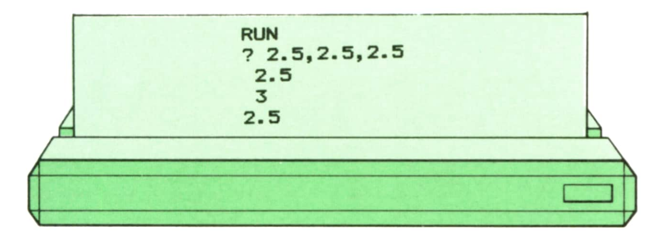
EN primer lugar, es posible leer los valores de varias variables con una sola instrucción INPUT. Para ello, basta poner sus nombres separados por comas:

INPUT variable, variable, variable.

Veamos un ejemplo idéntico al primer programa del capítulo anterior:



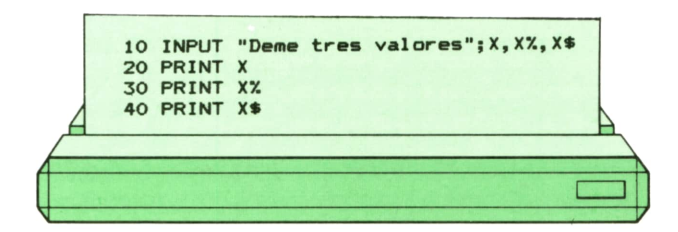
La única diferencia en la ejecución de este programa, respecto al que vimos en el capítulo anterior, es que el intérprete de BASIC espera recibir en una sola línea los valores de las tres variables cuyos nombres aparecen en la instrucción INPUT. Dichos valores deben estar separados por comas. Veamos cómo se ejecutaría esta nueva versión de nuestro programa:



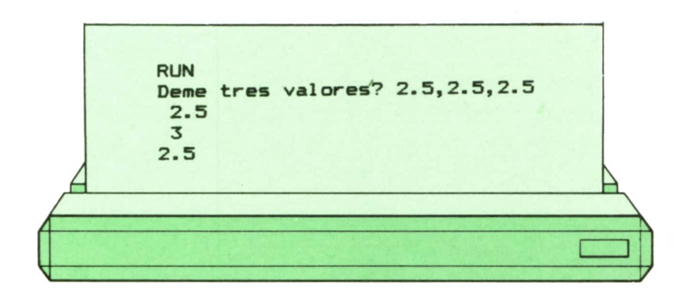
Como se ve, el resultado es el mismo, aunque los tres valores introducidos han sido escritos en una sola línea.

Otra modificación de la instrucción INPUT permite definir un mensaje que el intérprete escribirá en la pantalla antes de pedir los datos. Esto es muy útil si deseamos que el programa proporcione información utilizada sobre lo que espera de él. En un programa donde hay mucho intercambio de información entre el programa y la persona que lo utiliza, esta forma de la instrucción INPUT puede llegar a ser indispensable.

Veamos un ejemplo:



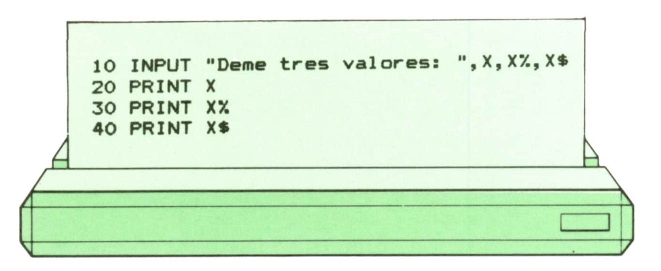
Veamos también su ejecución:



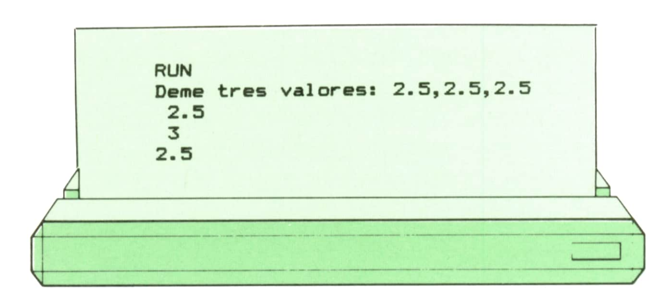
Como se ve, el mensaje se coloca delante de la lista de variables, separado de ellas por un punto y coma. Al ejecutarse, el mensaje aparece delante del signo de interrogación, en la misma línea.

Si queremos que el signo de interrogación desaparezca, bastará separar el

mensaje de la lista de variables por medio de una coma, en lugar de un punto y coma, como en el ejemplo siguiente:

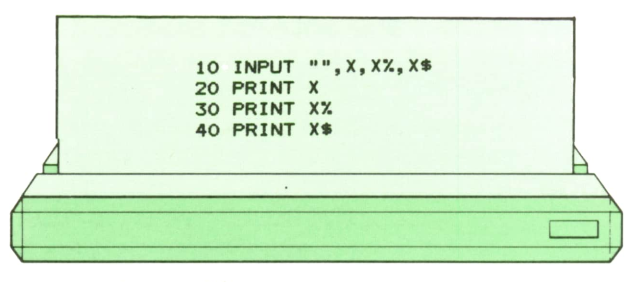


cuya ejecución es:

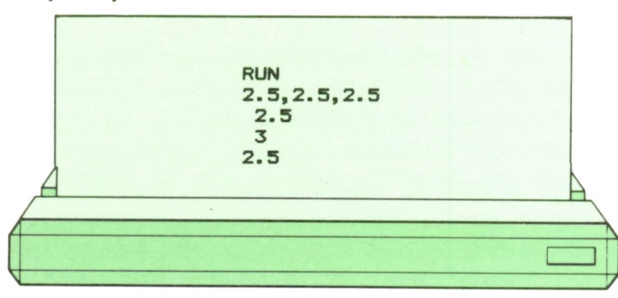


Obsérvese que, en este caso, para que aparezca un espacio en blanco de separación entre el mensaje y los datos introducidos, dicho espacio en blanco debe incluirse explícitamente dentro del mensaje.

Es evidente ahora cómo podría eliminarse el signo de interrogación sin sustituirlo por mensaje alguno. En efecto: bastará con utilizar un mensaje nulo (una cadena de caracteres sin ningún carácter, representada por dos dobles comillas seguidas). Veamos un ejemplo:



cuya ejecución es:



Escritura de una línea en la pantalla

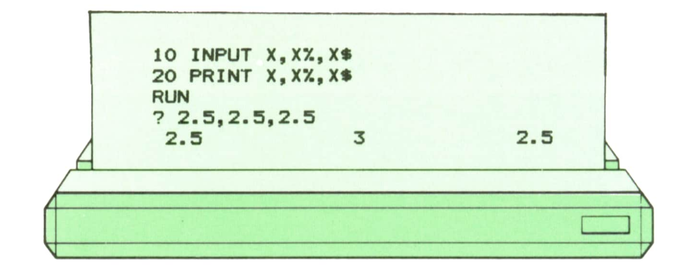
La instrucción BASIC inversa de la instrucción INPUT tiene la siguiente forma:

PRINT expresión

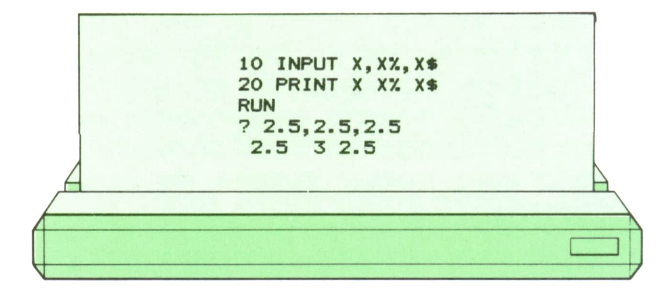
En los capítulos anteriores la hemos utilizado con frecuencia. Su efecto es hacer aparecer el valor de la expresión en la posición actual del cursor en la pantalla del ordenador. Todos los programas que hemos visto en este capítulo y en el anterior pueden servir como ejemplo más sencillo de su funcionamiento.

Obsérvese que PRINT puede aplicarse no sólo a una variable, sino también a una expresión, como en el último ejemplo del capítulo anterior.

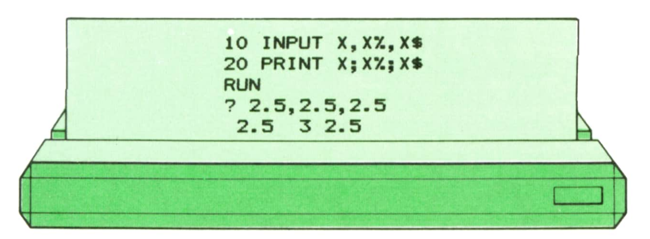
Al igual que en el caso de INPUT, es posible escribir en la pantalla el valor de más de una expresión, separándolas entre sí por espacios en blanco, comas o punto y coma. El efecto de cada uno de estos separadores es ligeramente diferente. Veamos diversos ejemplos, junto con sus ejecuciones correspondientes:



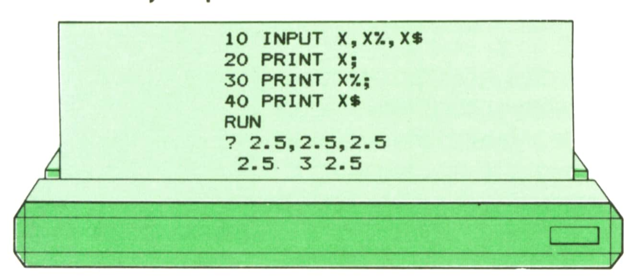
Cuando se utilizan comas como separadores, los valores de las expresiones quedan separados por cierto número de espacios en blanco.



Cuando se utilizan los espacios en blanco para separar las diversas expresiones, los resultados aparecen separados únicamente por uno o dos espacios en blanco.



Si se utiliza el punto y coma como separador, el resultado es idéntico al anterior. También puede colocarse un punto y coma al final de la línea, en cuyo caso quiere decir que no deseamos que el intérprete añada automáticamente un salto de línea el final del texto que acabamos de escribir con la instrucción PRINT. De esta manera, podremos concatenar en la misma línea los textos correspondientes a varias instrucciones PRINT. Veamos un ejemplo:

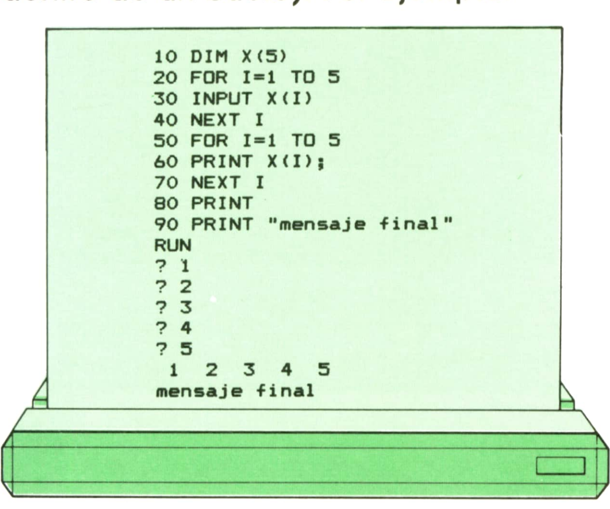


Se observará que el resultado obtenido es el mismo que al utilizar una sola instrucción PRINT.

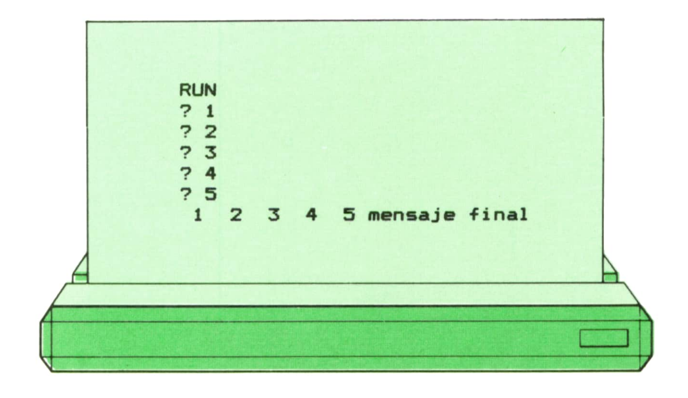
La instrucción

PRINT

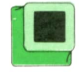
que no especifica ninguna variable, escribe sólo un salto de línea (pues no termina en punto y coma). Esta instrucción puede utilizarse para dar por terminada una línea que se ha ido generando sucesivamente (por ejemplo, con varias instrucciones PRINT, o con una sola situada dentro de un bucle). Por ejemplo:



mientras que, si elimináramos la instrucción 80 del programa anterior, obtendríamos:



porque no se habría producido el salto de línea, ya que la última instrucción PRINT ejecutada (la quinta vez que se ejecutó el bucle) terminó en punto y coma.



Entrada y salida por la consola en PASCAL

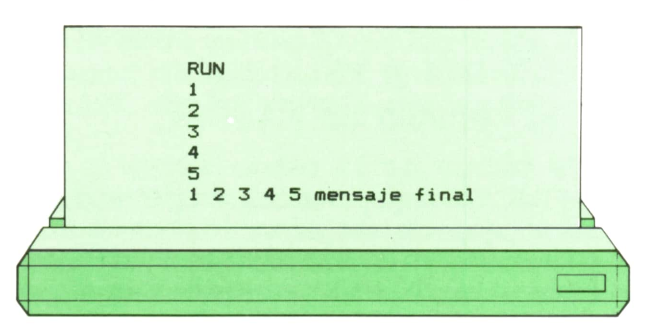
En PASCAL, para introducir datos por el teclado, puede utilizarse la instrucción READLN (en inglés «Read line», que quiere decir «leer línea»). Esta instrucción lee un dato del teclado y se lo asigna a la variable cuyo nombre se escribe entre paréntesis después de READLN, realizando las conversiones de tipo correspondientes, de una forma muy semejante a la instrucción INPUT de BASIC.

Para escribir en la pantalla, existen dos instrucciones principales: WRITE, que no añade automáticamente un salto de línea después de escribir (y corresponde, por tanto, a la instrucción PRINT de BASIC terminada en punto y coma) y WRITELN, que sí lo añade (y corresponde a PRINT cuando no termina en punto y coma). Tanto en el caso de WRITE como en el de WRITELN, lo que se escribe por la pantalla es el valor de la expresión o expresiones que se colocan entre paréntesis (separadas por comas) después de la palabra reservada WRITE o WRITELN.

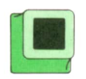
Veamos, como ejemplo, la versión en PASCAL del último programa que hemos escrito en BASIC, junto con su ejecución.

```
program ejemplo;
 var x:array[1..5] of integer;
     i:integer;
 begin
   for i:=1 to 5 do
       readln (x[i]);
   for i:=1 to 5 do
       write (x[i],' ');
   writeln ('');
   writeln ('mensaje final')
 end.
RUN
1
2
3
4
5
1 2 3 4 5
mensaje final
```

Obsérvese el papel de la instrucción WRITELN (''), que no hace otra cosa que escribir el salto de línea, pues la expresión entre paréntesis es una cadena de caracteres vacía. Si quitáramos la instrucción, el resultado del programa sería:

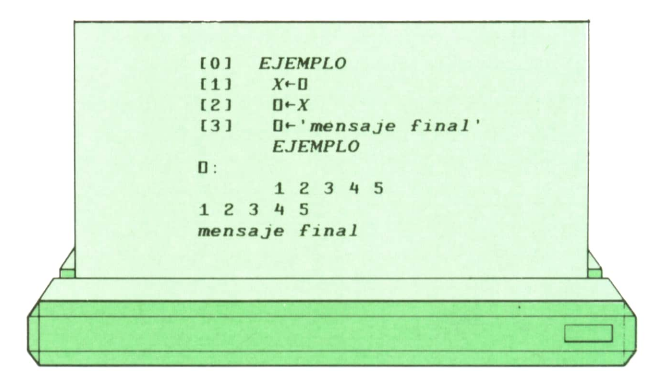


Obsérvese también que en PASCAL no se añaden automáticamente espacios en blanco de separación entre dos números consecutivos, por lo que hay que escribirlos explícitamente.

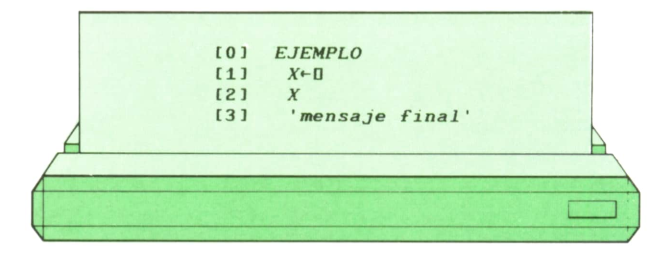


Entrada y salida por la consola en APL

En el lenguaje APL, la consola se considera como una variable más, que se representa con un cuadrado (que recuerda la pantalla). Esto significa que para realizar una operación de lectura, basta con preguntar el valor de dicha variable, mientras que para realizar una escritura en pantalla basta con asignarle un valor. Veamos el ejemplo que realiza el mismo papel que el que acabamos de ver en el lenguaje PASCAL, junto con su ejecución:



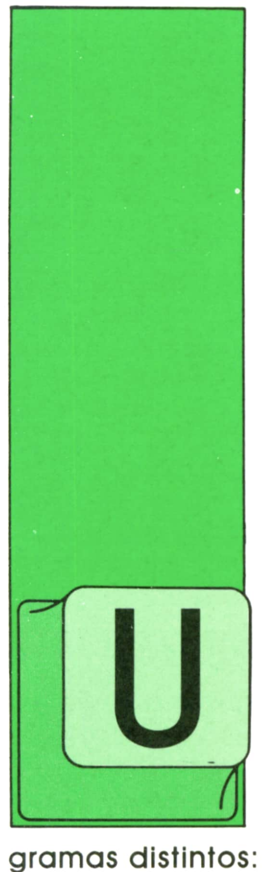
La asignación al cuadrado puede eliminarse si la última operación efectuada en esa línea no fue una asignación. Esto se cumple en nuestro caso, por lo que el programa podría escribirse también así:

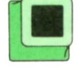


Además, si no se desea que se añada automáticamente un salto de línea al final, en lugar del cuadrado se escribe un símbolo distinto, un cuadrado superpuesto con una comilla:

```
[0]  EJEMPLO
[1]   X←⎕
[2]   ⍞←X
[3]   'mensaje final'
      EJEMPLO
⎕:
      1 2 3 4 5
1 2 3 4 5mensaje final
```

APLICACIONES





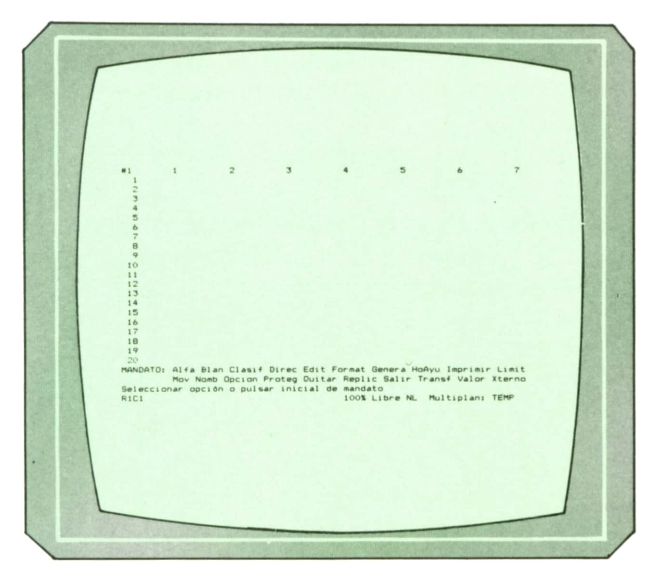
Paquetes integrados

UN paquete integrado es un conjunto de programas que facilita la realización de tareas complejas con el ordenador. Estos paquetes integrados incluyen cuatro programas distintos:

Procesador de textos.
Hoja de cálculo.
Gestor de base de datos.
Paquetes gráficos.

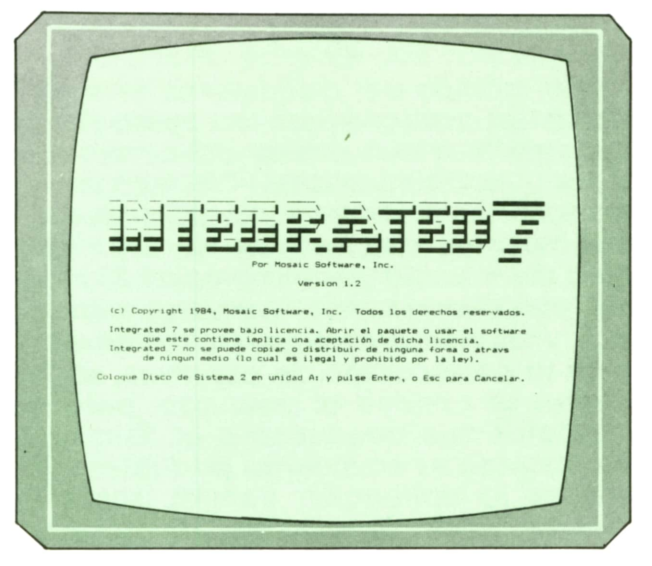
Existen muchos paquetes integrados que ofrecen además de estos programas otros tales como: agenda, calendario, comunicaciones, calculadora, etc.

En los programas normales, y salvo algún caso aislado, no era posible relacionarlos. Así tenemos paquetes de gráficos incapaces de interpretar los datos contenidos en un fichero, o procesadores de textos que no pueden comunicarse con la base de datos, es decir, eran absolutamente independientes unos de otros. Sin embargo, las tareas que realiza una empresa suele incluir normalmente procesos en los que son necesarios utilizar todos y cada uno de los programas anteriores, empleando además los mismos datos. Así, por ejemplo, se empezará a realizar unos determinados cálculos con la hoja electrónica, para después pasar a confeccionar los gráficos relativos a esos cálculos, resaltando de esta forma algunos aspectos importantes; posteriormente se elabora el informe correspondiente, incluyendo algún que otro gráfi-





Ejemplo de hoja de cálculo.



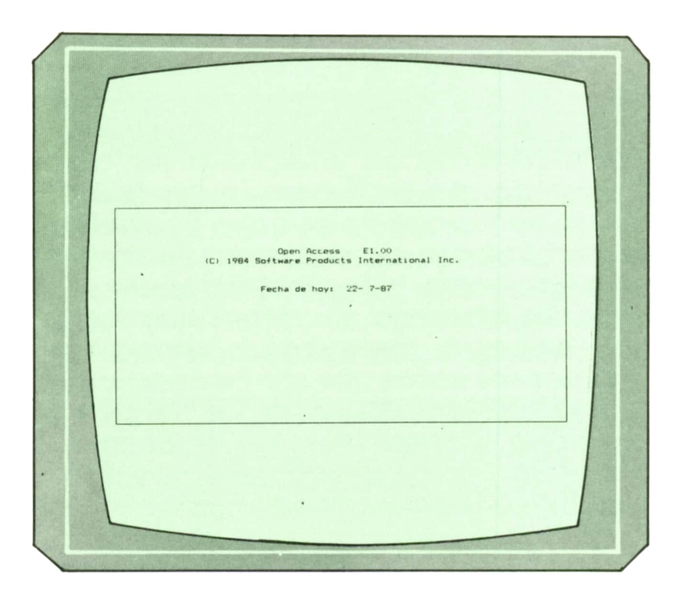


Paquete integrado integrated 7.

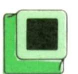
co; estos informes quizá haya que mandarlos a distintas personas, utilizando para ello los datos sobre sus direcciones, que figurarán en la base de datos.

Todas las tareas anteriormente dichas se vuelven dificultosas si tenemos que estar cambiando continuamente de programa, con el consiguiente aumento de la probabilidad de que se produzcan errores.

Estudiaremos ahora los distintos módulos que incluye un programa integrado.

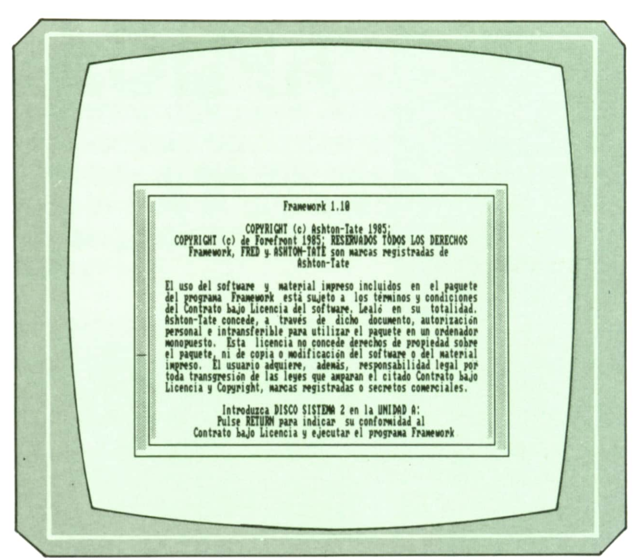



Pantalla de acceso a Open Access.



Hoja electrónica

Este módulo del paquete es el encargado de realizar todas las operaciones matemáticas necesarias, así como relacionar los distintos datos contenidos en los ficheros, efectuar comparaciones, ordenaciones, clasificaciones, etc.; una hoja de cálculo permitirá además realizar previsiones sobre los datos conocidos, y las más avanzadas permitirán también la persecución de objetivos, en los cuales se conoce el resultado, pero no los datos que conducen a él. Esta hoja electrónica es en muchos paquetes integrados la aplicación central, sobre la que se basan el resto de los módulos, siendo en algunos casos los otros módulos meras modificaciones de la hoja de cálculo.



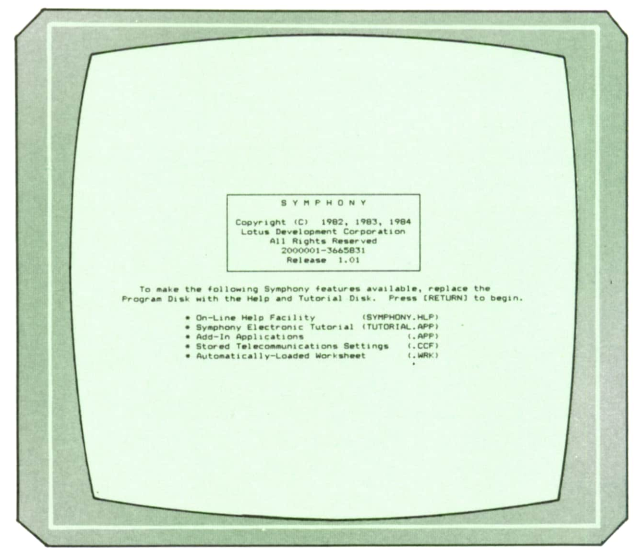

Pantalla de acceso a Framework.



Bases de datos

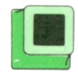
Este módulo permite la gestión de los datos contenidos en los ficheros, su recuperación, modificación, etc. Una de las características de esta base de datos es la facilidad con la que se modifica la estructura de los ficheros sin necesidad de alterar toda la estructura de la aplicación o tarea que se diseñó con el paquete integrado. El módulo incluye un lenguaje que facilita extraordinariamente la utilización de sus datos.

Los datos contenidos en los ficheros



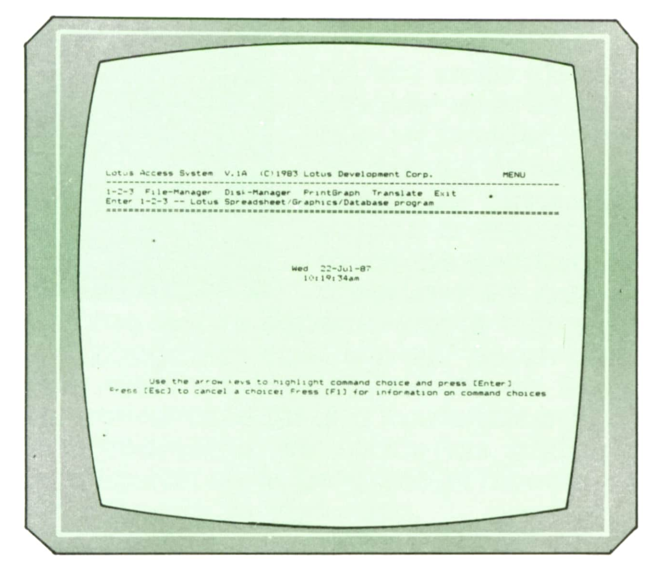

Pantalla de entrada en Symphony.

pueden ser utilizados por el resto de los módulos, especialmente por la hoja de cálculo.



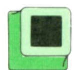
Gráficos

La función de este módulo es la de interpretar los datos contenidos en la base de datos o en la hoja de cálculo, y representarlos en forma de gráfico. Estos gráficos pueden ser de barras, de pastel o tarta, de líneas, etc., y de dos o tres dimensiones. El módulo es capaz de utilizar gran variedad de dispositivos de dibujo, desde simples impresoras con capacidades gráficas hasta sofisticados sistemas de dibujo llamados plotters.



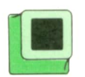


Pantalla de entrada en Lotus 1-2-3.



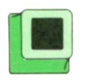
Comunicaciones

Es el módulo encargado de posibilitar la comunicación entre nuestro ordenador y otros exteriores, permitiendo de esta forma el traspaso de ficheros, datos, documentos, etc., de un ordenador a otro, o incluso con una red de ordenadores. Soportan tanto las comunicaciones síncronas como las asíncronas a través de un modem telefónico.



Agenda

Con este módulo podemos tener almacenados todos los teléfonos y direcciones útiles, siendo accesibles en cualquier momento. En algunos programas permite almacenar citas, de tal forma que advierte de la proximidad de una de ellas, en función de la fecha que se la suministre.



Justificación del software integrado

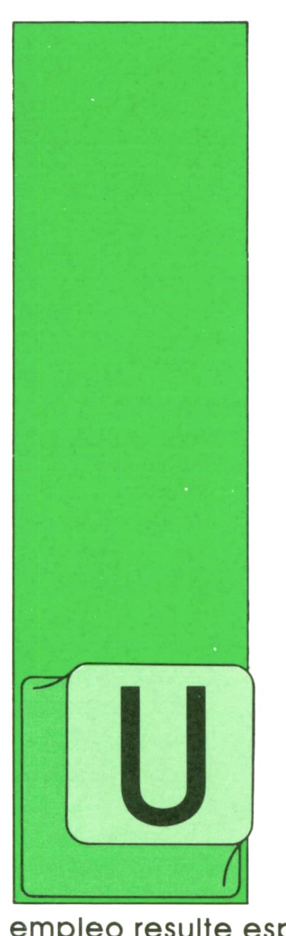
¿Cuáles son las razones por las que conviene utilizar un paquete integrado y no los programas sueltos?

En principio si los programas a utilizar no van a tener nada en común, resulta recomendable emplear programas independientes, ya que su precio es mucho menor, además de ofrecer más posibilidades al tener todos los recursos del ordenador dedicados a ellos.

Sin embargo, puede surgir un problema que decida utilizar un paquete integrado: la redundancia de los datos, esto es, tener datos repetidos que los utilicen en muchas aplicaciones, siendo prácticamente los mismos. Es aconsejable en este caso utilizar un paquete integrado que evitará tener ficheros que contengan los mismos datos.

En resumen, los paquetes integrados, que surgieron como una serie de programas interrelacionables, se han convertido en una de las más potentes herramientas de software del mercado, facilitando la programación de tareas sofisticadas que de otro modo serían de difícil mecanización.

PASCAL



Arboles de búsqueda (continuación)

UNA vez vistas las principales operaciones que se pueden realizar con los árboles binarios de búsqueda, vamos a realizar, a modo de ejemplo, un programa en que su empleo resulte especialmente indicado.

Supongamos que, a partir de un texto (por ejemplo, un capítulo de un libro), queramos obtener una lista ordenada alfabéticamente de todas las palabras que figuran en él junto con el número de apariciones de cada una.

En principio, podríamos pensar en alguna estructura de tipo Tabla donde ir guardando las sucesivas palabras que vayan apareciendo junto con un contador de apariciones para cada una. Con cada palabra que se fuese detectando en el texto habría que buscar primero en la tabla para ver si está ya registrada e incrementar su contador en caso afirmativo; si no hubiera aparecido con anterioridad, la nueva palabra quedaría anotada en la tabla y a su contador se le daría el valor uno. El hecho de tener que buscar en la tabla cada vez implica que, por motivos de eficiencia, la tabla debería estar ordenada alfabéticamente en todo momento, lo que, a su vez, obligaría a que, cada vez que se añadiese una nueva palabra, ésta se insertase en el lugar adecuado a base de desplazar un lugar todas las palabras posteriores y sus contadores para dejar hueco.

El hecho de que esta tarea pueda llegar a exigir mucho tiempo de ejecución, junto con la imposibilidad de saber con antelación el tamaño de tabla necesario, lleva a la conclusión de que una estructura de lista encadenada sería mejor para resolver nuestro problema, dadas las pocas operaciones necesarias para insertar un nuevo elemento y su característica de estructura dinámica.

Las listas encadenadas, sin embargo, tienen el inconveniente de que cada vez que se busca un elemento hay que empezar a explorar por el comienzo de la lista de elemento en elemento, con lo que, en promedio, el número de operaciones necesario puede ser muy alto.

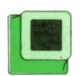
Dadas estas consideraciones, resulta, por tanto, que un árbol binario de búsqueda es una estructura de datos más adecuada para resolver nuestro problema que las tablas o las listas.

El tipo de nodo del árbol debería tener un campo para guardar la palabra correspondiente, campo que actuaría de «clave» a la hora de realizar las búsquedas; además, debería haber un campo numérico para el contador de apariciones y, lógicamente, dos campos de tipo puntero para apuntar a los descendentes. La definición de los tipos involucrados podría ser:

```
type
  Palabra_t = array [1..MaxLong] of char;
  Puntero_t = ^Nodo_t;
  Nodo_t      = record
                  Palabra       : Palabra_t;
                  Contador      : integer;
                  Izquierdo,
                  Derecho       : Puntero_t
                end;
```

donde MaxLong sería una constante definida con anterioridad.

El procedimiento de inserción en el árbol sería similar al que conocemos, con la particularidad de que, cuando la palabra ya estuviese anotada, la acción a ejecutar sería la de incrementar el contador correspondiente. Veamos a continuación cómo resolver las otras partes del programa.



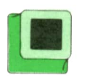
Extracción de palabras del texto

Vamos a suponer que el texto se encuentra almacenado en un fichero de disco. Tras abrir el fichero (cosa que en el programa haremos según las normas del Turbo-Pascal), iremos leyendo las letras del texto de una en una.

En un momento dado nos podremos encontrar en dos situaciones distintas: en fase de componer una palabra o avanzando entre dos palabras consecutivas.

Si no nos encontramos en fase de componer una palabra, la única comprobación que hay que hacer tras leer cada letra es ver si hemos llegado al comienzo de una nueva palabra, es decir, ver si aquélla pertenece al conjunto de caracteres admitidos como posibles letras iniciales; en caso afirmativo, la letra pasaría a ser la primera de una nueva palabra y entraríamos en fase de composición.

Estando en fase de composición, la comprobación que hay que realizar para cada letra es la de ver si está incluida o no en el conjunto de caracteres admitidos como posibles integrantes de una palabra (conjunto que no tiene por qué coincidir con el de iniciales). Si la letra es aceptable, hay que añadirla a la nueva palabra en fase de composición (comprobando antes que no se excede la máxima longitud prevista para palabras en el programa); en caso contrario, está claro que el proceso de composición ha terminado y, en nuestro caso, hay que guardar la nueva palabra en el árbol. Para facilitar posteriormente las comparaciones entre palabras conviene, a medida que se van ensamblando, pasar las letras a minúsculas.



Presentación de resultados

Para presentar los nodos de un árbol de búsqueda de manera que aparezcan ordenados según el criterio de comparación utilizado, basta con recorrerlo recursivamente en orden central. En efecto, antes de presentar el contenido de un nodo hay que presentar todos los de su subárbol izquierdo, pues se encuentran por delante; análogamente, sólo tras presentar el contenido de un nodo se puede pasar a representar el contenido de los de su subárbol derecho, al encontrarse éstos por detrás. Al tiempo que se recorre el árbol es posible llevar la cuenta del total de palabras extraídas (suma de los contadores de todos los nodos) y del número de palabras distintas encontradas (número de nodos).

Llevadas a la práctica todas estas consideraciones, tendríamos el siguiente programa:

```
program Lexico;
(*---------------------------------------------------------*)
(* Saca una lista de las diferentes palabras que aparecen *)
(* en un texto, junto con sus respectivas frecuencias.    *)
(*---------------------------------------------------------*)
const
 MaxLong = 30;   (* Máximo número de caracteres por palabra *)

type
 Palabra_t = array [1..MaxLong] of char;
 Puntero_t = ^Nodo_t;
 Nodo_t    = record
              Palabra    : Palabra_t;
              Contador   : integer;
              Izquierdo,
              Derecho    : Puntero_t
             end;

 (* Posibles resultados de la comparación de dos palabras: *)
 Test_t    = (Antes, Iguales, Despues);
```

```
var
 Validas,
 Iniciales : set of char;    (* conjuntos de letras aceptables    *)

 Raiz     : Puntero_t;
 Fichero : text;
 Nombre,                     (* nombre del fichero de texto       *)
 Nueva    : Palabra_t;       (* palabra en fase de composición    *)
 Letra    : char;            (* letra recién leída                *)
 EnMedio : boolean;          (* ¿estamos en medio de una palabra? *)

 Longitud,                   (* longitud de Nueva                 *)
 I,
 Diferentes,                 (* contador de palabras diferentes   *)
 TotalPalabras: integer;     (* contador del total de palabras    *)

(*-----------------------------------------------------------------------*)
function Test (var A,B: Palabra_t): Test_t;
(*----------------------------------------------------*)
(* Compara dos palabras alfabéticamente; puede devolver *)
(* los resultados Antes, Igual o Despues               *)
(*----------------------------------------------------*)
  var I: integer; Distintas: boolean;
begin
 Distintas:= false;
 I:= 1;
 while (I <= MaxLong) and not Distintas do
   if A[I] <> B[I] then Distintas:= true
                   else I:= I+1;

 if Distintas then if A[I] < B[I] then Test:= Antes
                                  else Test:= Despues
              else Test:= Iguales
end;

(*-----------------------------------------------------------------------*)
procedure InsertarEn (var P: Puntero_t);
 begin
 if P = nil then
     begin                  (* crear nueva ficha *)
      new (P);
      with P^ do
        begin
         Palabra   := Nueva;
         Contador  := 1;
         Izquierdo:= nil;
         Derecho   := nil
        end
     end
  else
     with P^ do
       begin
        case Test (Nueva, Palabra) of
          Iguales: Contador:= Contador + 1;
          Antes   : InsertarEn (Izquierdo);
          Despues: InsertarEn (Derecho)
         end
       end
 end;
(*-----------------------------------------------------------------------*)
procedure Muestra_Arbol (P: Puntero_t);
(*----------------------------------------------------------------*)
(* Recorre y muestra el árbol apuntado por P en orden central, *)
(* es decir, en orden alfabético. Además, lleva la cuenta de   *)
(* palabras distintas y del total de palabras.                 *)
(*----------------------------------------------------------------*)

 begin
  if P <> nil then with P^ do
    begin
     Muestra_Arbol (Izquierdo);

     write (Palabra); writeln (Contador:6);
     TotalPalabras:= TotalPalabras + Contador;
     Diferentes    := Diferentes + 1;

     Muestra_Arbol (Derecho)
    end
 end;
(*-----------------------------------------------------------------------*)
function Minuscula (L: char): char;
(*------------------------------------------*)
(* Devuelve la letra L en minúscula, si procede *)
(*------------------------------------------*)
 begin
```

```
   case L of
     'A'..'Z': Minuscula:= chr (ord(L) - ord('A') + ord('a'));
     'Ñ':      Minuscula:= 'ñ';
     'É':      Minuscula:= 'é';
     'Ü':      Minuscula:= 'ü';
     else      Minuscula:= L
    end
  end;

(*---------------------------------------------------------------------------*)
begin
  Iniciales := ['a'..'z','ñ','á','é','í','ó','ú'];
  Validas   := ['a'..'z','ñ','á','é','í','ó','ú','0'..'9','_','ü'];

  Raiz    := nil;
  Enmedio := false;

  write  ('Nombre del fichero: ');
  readln (Nombre);
  assign (Fichero, Nombre);      (* Específico del Turbo-Pascal *)
  reset  (Fichero);

  while not eof(Fichero) do
   begin
    read (Fichero,Letra); Letra:= Minuscula (Letra);

    if not EnMedio then
        begin
         if Letra in Iniciales then
           begin
            EnMedio  := true;
            Longitud := 1;
            Nueva [1]:= Letra
           end
        end     (* par begin/end aparentemente innecesario...*)

     else
       if Letra in Validas then
           begin
            if Longitud < MaxLong then
              begin
               Longitud        := Longitud + 1;
               Nueva [Longitud]:= Letra
              end
           end (* idem...*)

         else           (* se acabó una palabra: *)
            begin
             Enmedio:= false;    (* completa Nueva con blancos: *)
             for I:= Longitud + 1 to MaxLong do Nueva [I]:= ' ';
             InsertarEn (Raiz);
            end
    end;

  close (Fichero);        (* Específico del Turbo-Pascal *)

  ClrScr;
  writeln ('Lista por orden alfabético:'); writeln;
  Diferentes:= 0; TotalPalabras:= 0;
  Muestra_Arbol (Raiz);
  writeln;
  writeln ('Palabras: ',Diferentes,'   Total: ',TotalPalabras);

end.
```

Como en otras ocasiones, se ha supuesto que el compilador disponible es el Turbo-Pascal a la hora de programar la apertura y cierre del fichero.

En el procedimiento InsertarEn, en lugar de pasar en la lista de parámetros el elemento a insertar, se ha utilizado directamente la variable global Nueva, a fin de

ahorrar tiempo y espacio de memoria.

El programa, tal como está, tiene algunas imperfecciones:

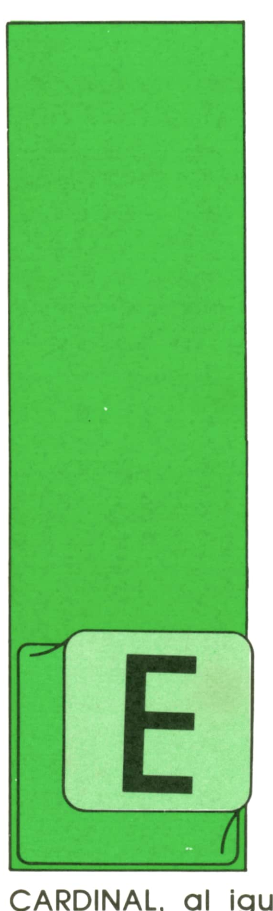
— No se detectan las palabras repartidas en dos líneas por medio de guión.

— Las vocales acentuadas, a la hora de clasificar palabras se consideran distintas a las no acentuadas y posteriores a la Z.

— No se detectan posibles errores en el acceso a los ficheros.

OTROS LENGUAJES

(MODULA-2 2)



EL lenguaje es muy similar al PASCAL, por lo que nos limitaremos a indicar las principales diferencias.

Existen dos nuevos tipos básicos de datos CARDINAL y BITSET. CARDINAL, al igual que INTEGER, representa número entero positivo. Se aplican los mismos operadores que a los enteros (+ para la suma, – para la resta, * para la multiplicación, DIV para división entera y MOD para el resto de la división).

Las ventajas de usar CARDINAL en vez de INTEGER están en el hecho de declarar explícitamente que sólo se admiten valores positivos. De esta forma si se intenta asignar, como constante o como resultado de una operación, un número negativo el programa emitirá un mensaje de error y detendrá su ejecución en vez de continuar, con los posibles resultados negativos para el funcionamiento del programa.

Otra ventaja adicional es que por cuestiones de representación interna de los datos las operaciones de multiplicación y la división son ligeramente más rápidas que si utilizáramos el tipo INTEGER, con el consiguiente ahorro de tiempo al realizar un gran número de estas operaciones.

MODULA también permite expresiones mixtas en las que intervengan operadores de ambos tipos. En este caso deberá utilizarse funciones de transferencia de tipo. Si *i* es de tipo INTEGER y *c* de tipo CARDINAL, la expresión *i* + *c* sería errónea, pero *i* + INTEGER *(c)* sería de tipo INTEGER, y *c* + CARDINAL *(i)* del tipo CARDINAL.

BITSET es un conjunto de enteros entre 0 y N – 1, donde N es una constante definida por el compilador usado. Normalmente N coincide con la longitud de la palabra del ordenador utilizado, o de submúltiplo de ésta. Su principal utilidad se verá en el apartado dedicado a las funciones de bajo nivel.

Las estructuras de control son muy similares a las del PASCAL, excepto que se ha eliminado la palabra clave BEGIN. El comienzo se indica por la sentencia de la estructura y el final por END.

Por ejemplo, una estructura del tipo WHILE se escribiría de la siguiente forma:

```
WHILE i < 0 DO
    Z := Z * X;
    i := i + 1;
END
```

La estructura REPEAT funciona igual que en PASCAL.

```
REPEAT
    Z := Z * X;
    i := i + 1;
UNTIL i >0
```

Para bucles infinitos existe la sentencia LOOP. Esta realiza un bucle hasta que se ejecuta la instrucción EXIT que lo finaliza. La ventaja es que la instrucción EXIT puede estar colocada en cualquier parte dentro del bucle, no siendo necesario completarlo para salir de él.

```
LOOP
   IF i = 5 THEN EXIT;
   i := i + 1;
END
```

La instrucción FOR incluye una parte para indicar el paso con el que debe realizarse el bucle, igual que en el STEP del BASIC.

```
FOR i := 5 to 20 BY 3 DO
      j := j + a(i)
END
```

Incrementará la variable *i* de 3 en 3.

En la instrucción IF se ha incluido, aparte del ELSE, la instrucción ELSEIF, que permite anidar diversas estructuras de este tipo sin problemas de a quién pertenece cada ELSE, como surgen en algunos programas de PASCAL.

```
IF i < 0 THEN
       mayor;
ELSEIF i > 0 THEN
       menor;
ELSEIF i = 0 THEN
        igual
END
```

En la sentencia CASE se ha incluido la parte ELSE, que se ejecuta sin ninguna de las otras selecciones ocurre, lo cual era uno de los defectos del PASCAL.

El manejo de los arrays, o matrices, se ha hecho más flexible, de esta forma se puede declarar matrices de un tipo sin indicar sus dimensiones, con lo que se pueden pasar como parámetros a procedimientos, simplificando así las cosas, por ejemplo, al diseñar diversos algoritmos de ordenador, para los que antes había que realizar uno por cada matriz.

También es más flexible la declaración de constantes, ya que ahora se permiten declaraciones compuestas que permitan funciones entre los operandos.

```
CONST
   MAX = 5;
   SITIO = (MAX*5)+2;
```

En cuanto los punteros, que son estructuras dinámicas de datos, con las que se pueden crear, o destruir según nos interese, diversos tipos de datos.

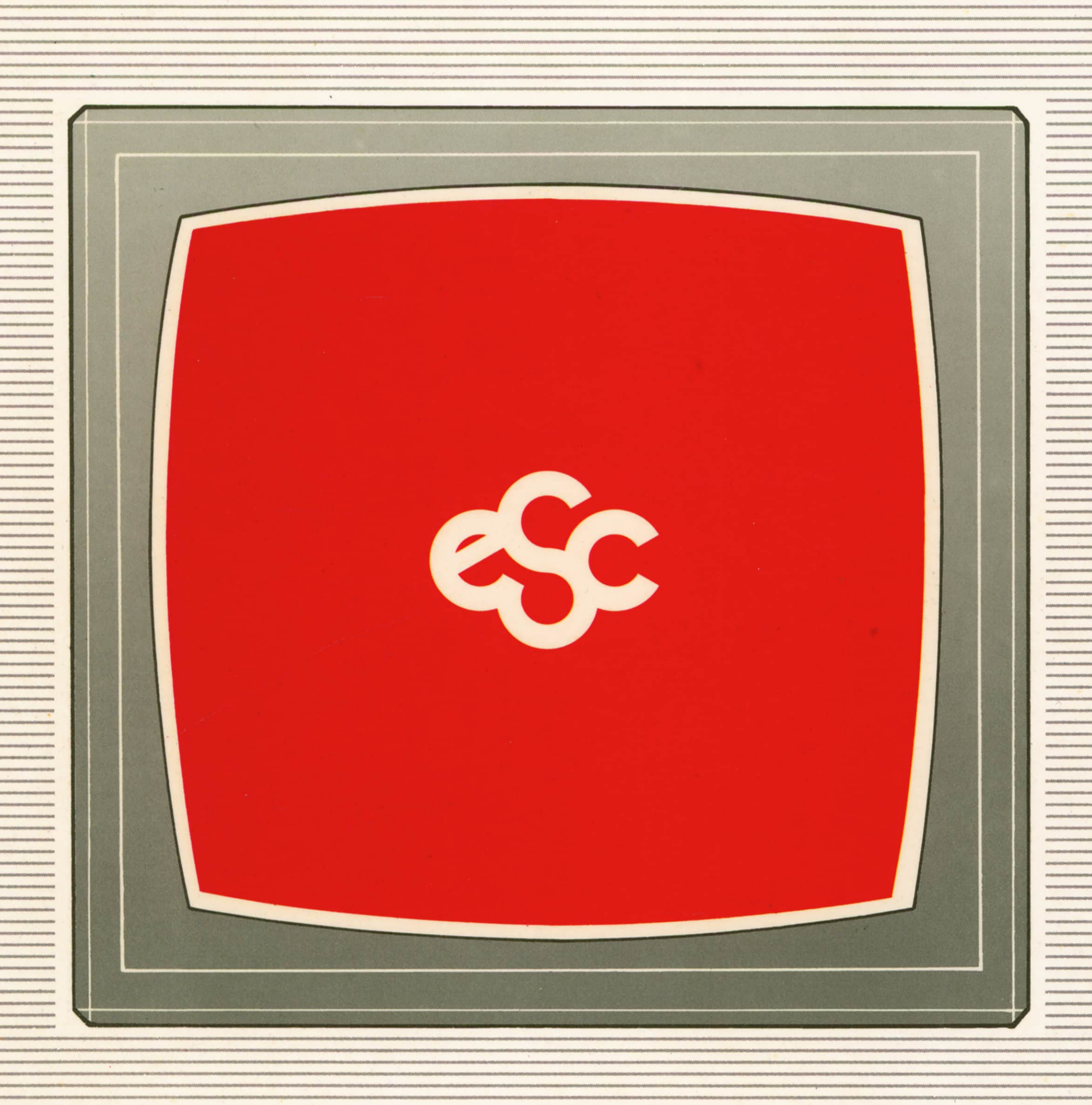
Estos se creaban o destruían en PASCAL mediante las sentencias NEW y DISPOSE. En MODULA las equivalentes son:

```
NEW (p) = ALLOCATE (p,TSIZE(T))
DISPOSE (p) = DEALLOCATE (p,TSIZE(T))
```

Estas funciones pertenecen al módulo Storage. Por tanto, antes de utilizarse deberemos incluir en el encabezamiento del módulo:

FROM Storage IMPORT ALLOCATE, DEALLO CATE;

El puntero vacío continúa siendo NIL.



▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼